The University Of Sheffield.

DEPARTMENT OF ELECTRONIC AND ELECTRICAL ENGINEERING

Autumn Semester 2007-2008 (2 hours)

Answers to Advanced Computer Architectures 4

1. a. *A method for producing the sine/cosine of an arbitrary angle using only trivial multiplies (shifts) can be based on an implementation of the CORDIC (**co**-ordinate **r**otation **di**gital **c**omputer) algorithm. The algorithm begins with two values $x_0 = 1$ and $y_0 = 0$, an initial angle, $\beta_0 = 0$, and an angle, θ, and iterates towards the final values of $x_N = \cos\theta/K$, $y_N = \sin\theta/K$, and $\beta_N = \theta$ as follows:*

$$x_{i+1} = x_i - S_i 2^{-i} y_i$$
$$y_{i+1} = S_i 2^{-i} x_i + y_i$$
$$\beta_{i+1} = \beta_i + S_i \gamma_i$$

where $S_i = 1$ *when* $\theta - \beta_i > 0$,
$= -1$ *when* $\theta - \beta_i < 0$

and $\gamma_i = \arctan(2^{-i})$

*Somebody had devised some simple hardware to implement the algorithm, as shown in **Figure 1**.*

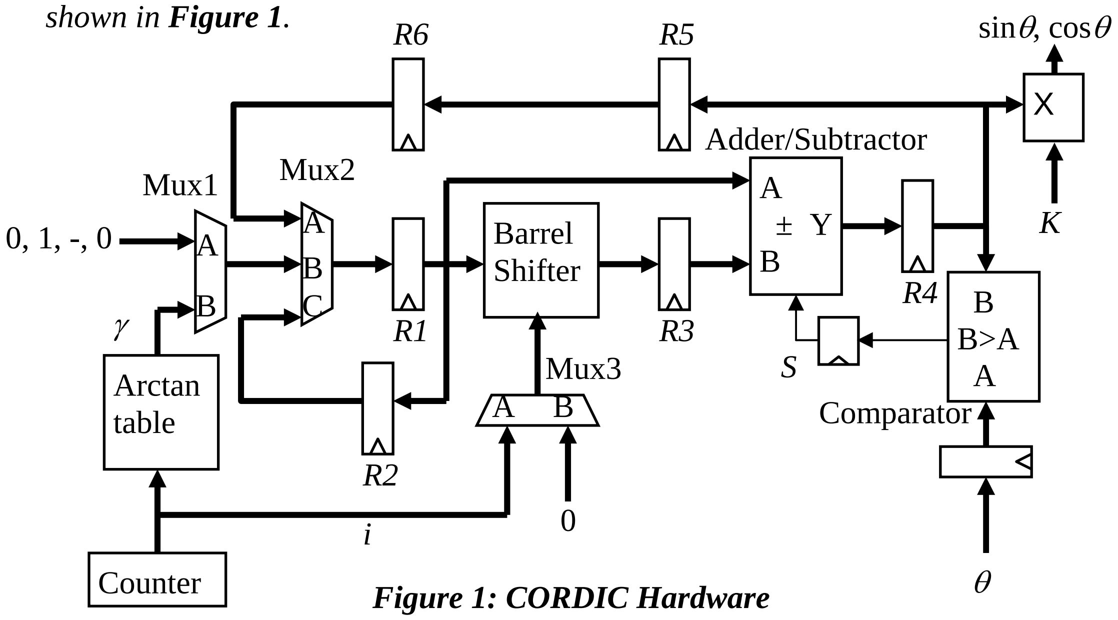


Figure 1: CORDIC Hardware

The counter provides the value of i that is used to control the barrel shifter (which can provide a controllable shift of the data equivalent to multiplying it by 2^{-i}) during each iteration and uses a table to look-up the value of $\arctan(2^{-i})$. During each iteration, at the appropriate point, the comparator is used to compare the value of β_{i+1} (calculated during iteration i) with θ to set the value of

S_{i+1} for the next iteration. The value of S controls the adder/subtractor, which can perform A+B, A-B, and B-A, to add or subtract as required.

All that the designer tells you is that the data moves, word-serially, around the system in the following order: y_i, x_i, -, β_i, γ_i (- denotes no specific value) and that the initial conditions are set by inserting the values 0,1,-,0,arctan(1) at the left hand side whilst the initial value of S is set to 1.

The final output is provided by multiplying x_N, and y_N by K (=0.60725) after the final iteration.

i. *Show how the algorithm is mapped onto the hardware by producing a reservation table for the first few iterations of the algorithm. Ensure that you identify the state of the two multiplexers at each clock cycle and the operation performed by the adder/subtractor.*

This is actually simpler than it looks and the key to doing this is compare the sequence of the datum (the order in which they are presented) with the requirements of the algorithm. The information that is being asked for is a reservation table with some additional information about the states of the two multiplexers and the operation. To facilitate this, the registers have been named: R1 to R6.

i	R_1	R_2	R_3	R_4	R_5	R_6	Mux1	Mux2	Mux3	Op
1	y_0						A	B		-
1	x_0	y_0	$y_0 2^{-1}$				A	B	A	-
1	y_0	x_0	$x_0 2^{-1}$	$x_0\ nS\ y_0 2^{-1} = y_1$			-	C	A	A *nS* B
1	β_0	y_0	$y_0 2^{-1}$	$y_0\ S\ x_0 2^{-1} = x_1$	y_1		A	B	A	B *S* A
1	γ_0	β_0	$\beta_0 2^{-0}$	-	x_1	y_1	B	B	B	-
1	y_1			$\beta_0\ S\ \gamma_0 = \beta_1$	-	x_1	-	A	A	B *S* A
2	x_1	y_1	$y_1 2^{-2}$		β_1		-	A	A	-
2	y_1	x_1	$x_1 2^{-2}$	$x_1\ nS\ y_1 2^{-2} = y_2$		β_1	-	C	A	A *nS* B
2	β_1	y_1	$y_1 2^{-2}$	$y_1\ S\ x_1 2^{-2} = x_2$	y_2		-	A	A	B *S* A
2	γ_1	β_1	$\beta_0 2^{-0}$	-	x_2	y_2	B	B	B	-
2	y_2			$\beta_1\ S\ \gamma_1 = \beta_2$	-	x_2	-	A	A	B *S* A
3	x_2	y_2	$y_2 2^{-3}$	-	β_2	-	-	A	A	-
3	y_2	x_2	$x_2 2^{-3}$	$x_2\ nS\ y_2 2^{-3} = y_3$		β_2		C	A	A *nS* B

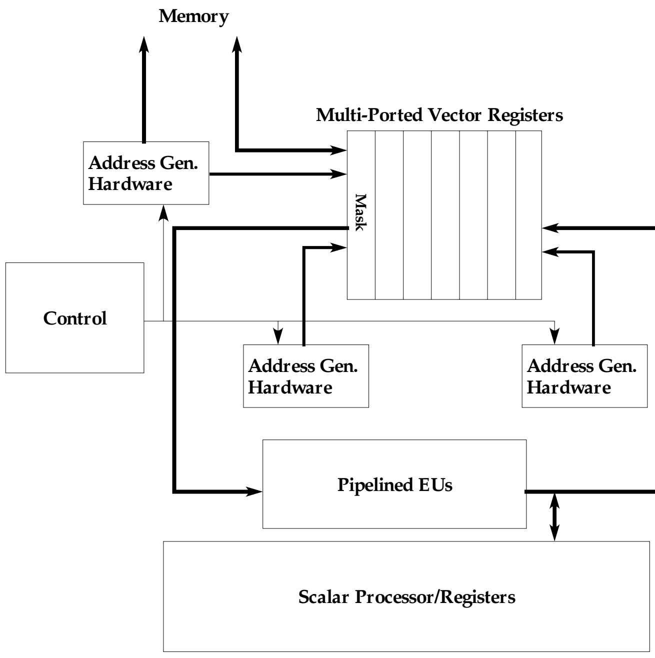
Where *S* implies + when true, - when false and *nS* implies – when true, + when false. Furthermore, the new value of *S* for the next iteration is set at the end of the clock cycle when β is in R4. **(15)**

ii. *You know that 40 iterations will produce the correct result but you want to improve the design so that the result will be output in the minimum number of iterations: how might this be done?*

You know that you are approaching the correct angle as β approaches θ. Consequently, if the comparator also detected equality of β to θ to some arbitrary precision then the hardware could signal when the operation was complete. **(5)**

2. a. *Describe the organisation of a vector processor and how it differs from a scalar processor. Ensure that your answer includes a diagram identifying the various parts clearly and the role that they play.*

A vector processor executed operations on vector values (n-tuples) rather the individual scalar values. Thus, a multiply operation would multiply two vectors together on an element by element basis to yield a new vector of the same size. The processing elements of a vector processor can be highly pipelined because **(6)** the vectors will usually be long enough to ensure that the latency incurred in these pipelines is small (on a per datum basis). Similarly, the memory interface will be organised to transfer vectors between memory and the processor. The processor will probably contain vector registers which can be organised into particular lengths, e.g. 16 x 256, 8 x 512, etc. A vector processing system will probably contain a scalar processor to allow scalar operations to be accomplished efficiently.



b. **i.** *What problems are encountered when loading vectors from multi-dimensional arrays held in memory.*

Two (or more) dimensional vectors (or matrices) are organised in memory as a contiguous list of 1 dimensional vectors. For example row-ordered implies that the data in each row are held in contiguous memory locations and the row vectors are stored contiguously. However, this organisation, whilst making it easy to access row vectors, makes it difficult to access column vectors. **(4)**

ii. *How might these problems be addressed.*

Fast memory accessing modes can be used to speed up transfers of row vectors, e.g. using page-mode in DRAMs and low-order interleaving, but each datum from a column must be fetched individually incurring a longer transfer time. This effect can be ameliorated by organising the memory into n medium-order interleaved banks such that the value retrieved from each bank is from the same column. By these means, the column accessing can be speeded-up by a factor approaching n (depending on the connection between memory and the processor) at the expense of added complexity. Unfortunately, this method will only support particular strides. **(4)**

d. *Which of the following code snippets would transfer easily onto a vector processor and why would this be the case.*

i. `for (i=0; i<N; i++) z[i]=x[i)*y[i];`

This one is OK. The array being written to does not appear on the RHS and the loop might equate to one or more vector multiplies **(2)**

ii. `for (i=L; i<N; i++) z[i]=x[i]*z[i-L];`

This one will be OK for a critical value of L. To faithfully mirror the action of this code snippet, the vectorised code must use the values of z already computed within the loop to compute new values of z. Due to the deep pipelining used in vector execution units, the index of z written as z[i] is being read from the vector registers will have an index less than i (difference related to the pipelining depth). If L is big enough then the value will have been written by the time it is read back. **(2)**

iii. `for (i=0; i<(N-1); i++) z[i]=x[i]*z[i];`

This one will be OK. To faithfully mirror the action of this code snippet, the vectorised code must use the original values of z on the RHS. Consequently, this can be read because it will not be overwritten until well after it is read. **(2)**

where `N` *is the length of the array and* `L` *is a positive integer that is less than* `N`

3. *A set of 32 processors is interconnected via a Cross-Point-Switch as shown in* ***Figure 3****.*

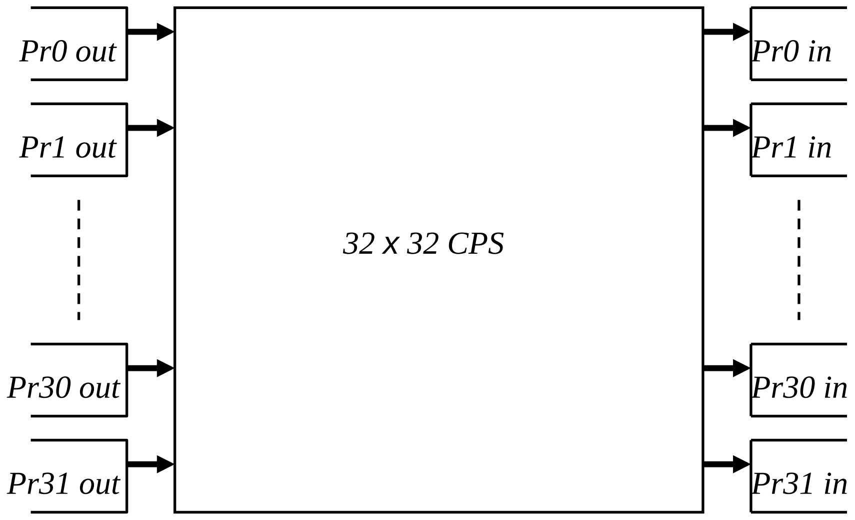


Figure 3: Processor Organisation

The network of processors is running an application where each processor sends messages to all of the other processors at a rate of approximately 300,000 messages per second (you can assume that the messages are evenly distributed and that the number of processors is large enough so that the fact that a processor does not send messages to itself does not affect the answer). The time taken to send each message (without contention) is 1.5μs.

a. *Given that the situation in* ***Figure 3*** *resembles the case of a set of processors connected to shared memory:*

i. *state the expression that gives the probability, p_A, that each message sent by a processor will be accepted at its destination without contending with another message, with an explanation of the terms;*

$$p_A = \frac{M}{NRT_{mess}}\left(1-\left(1-\frac{RT_{mess}}{M}\right)^N\right)$$

(4)

Where:

N is the number of input ports to the CPS,
M is the number of output ports from the CPS,
R is the rate at which a processor is generating messages,

ii. calculate the probability for this case.

$N=M=32$, $R=3\text{x}10^5$, $T_{mess}=1.5\text{x}10^{-6}$. $p_A = 0.8$ **(2)**

b. *In the event of contention, the processor will wait for 1.5μs and try again. On this basis, calculate the approximate, average time taken to send each message.*

The message will get accepted at the first attempt with a probability p_A and this will contribute a time of p_AT_{mess}. It fails to get accepted first time with a probability of (1-p_A) (incurring a delay of T_{mess}) and then gets accepted second time with a probability of p_A, again. This will contribute a time of 2(1-p_A) p_AT_{mess}. This will continue with each retry incurring an additional cost of T_{mess} with a decreasing probability. I will be prepared to accept an approximate answer based on a manual sum of the first few terms.

However, overall the exact average time incurred is, assuming that $x = (1 - p_A)$:

$$T_{ave} = T_{mess} p_A \sum_{i=1}^{\infty} i(1 - p_A)^{i-1}$$
$$= T_{mess}(1 - x)\sum_{i=1}^{\infty} i.x^{i-1}$$

Now, $\int \sum_{i=1}^{\infty} i.x^{i-1} dx = \sum_{i=1}^{\infty} x^i + K = 1 - \frac{1}{1-x} + K$

and so, $\sum_{i=1}^{\infty} i.x^{i-1} = \frac{d}{dx}\left(1 - \frac{1}{1-x} + K\right) = \frac{1}{(1-x)^2}$

Thus,

$$T_{ave} = T_{mess}(1 - x)\sum_{i=1}^{\infty} i.x^{i-1}$$
$$= \frac{T_{mess}(1-x)}{(1-x)^2} = \frac{T_{mess}}{1-x} = \frac{T_{mess}}{p_A} = 1.875\mu s$$

(6)

c. *The application is changed such that all of the processors continue to send messages at the same rate but directed (equally) only to processors Pr0…Pr3.*

i. *How does this affect the probability p_A and the average time taken to send each message?*

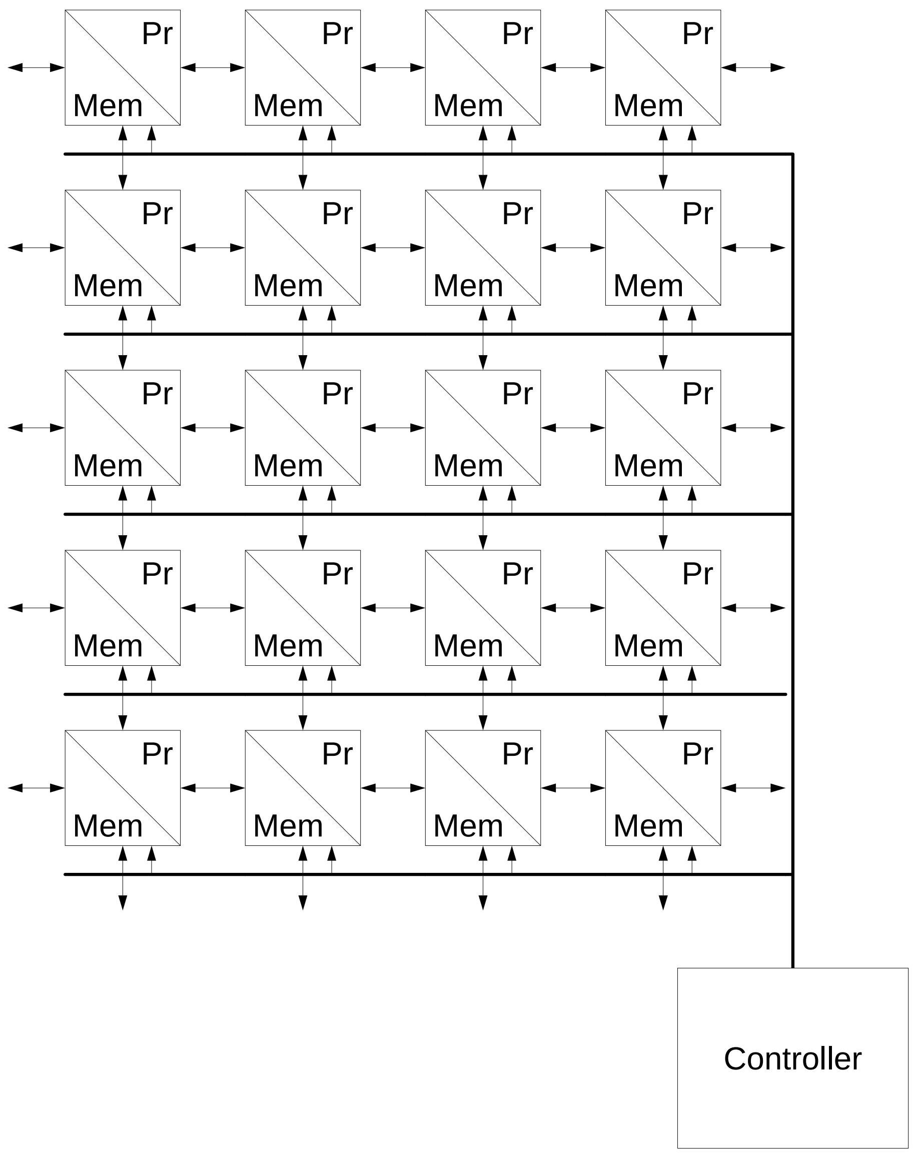
In this case, the CPS now looks like 32 inputs but only 4 outputs (N=32, M=4). Consequently, the value of p_A is now 0.27 and this now means that the average time to send a message is now 5.55µs. **(3)**

ii. *What would the consequence of this change be?*

The problem with this is that each processor could only send 180,000 messages per second and this is lower than the 300,000 that it is assumed that each processor is sending. Consequently, the rate at which the processors are sending message will be lower – but not 180,000 because if this were the rate of sending messages, the rate that could be supported would be 275,000. In fact the equilibrium point will be at approximately 230,000 messages per second. **(5)**

4. **a.** *Identify the important features of a SIMD array processor. Ensure that your explanation identifies how such a processor could be described as SIMD.*

A diagram such as this one should be included:



The written answer should include:

1,2, or more bit processors,

memory between 64-1024 bits,

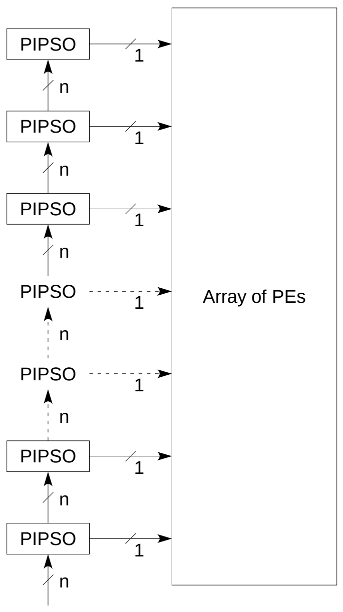
usually 2D interconnection

centralised control (i.e. single instruction on multiple data i.e. SIMD)

Additional logic to make loading/unloading data create a smaller overhead

Spatial invariance of operations/conditional operation. **(6)**

What is the purpose of a corner-turn buffer? How is it used?



The CTB is, in effect, a 2-dimensional shift register as shown here. A word-oriented column of data is loaded through the parallel inputs (in a word-serial manner) of the Parallel Input, Parallel/Serial Output shift-registers until the CTB contains a whole column of data. At this point, the individual words are shifted into the array of PEs from the serial outputs of the PIPSO registers simultaneously. This operation is repeated for every column of data. The first time it is done, the data is shifted into the first column of PEs. The second time, the first column of data is shifted into the second column, and so on. Obviously, data can be unloaded from the array using a similar mechanism.

(4)

An n `x` *n array can support the following operations:*

```
MOVE   addr1,addr2          ; (addr1) → (addr2)

CSWAP  addr1,addr2          ; if (addr1) > (addr2) then (addr1) ↔ (addr2)

BCAST  addr                 ; (addr) →N,S,E, and W link

IN     x,addr               ; x (=N,S,E, or W) → (addr)
```

An n `x` *n array of data, f, is distributed across the array (being placed in memory location 0 of each processor). Write a program to calculate the value of g in the following equation:*

$$g(x,y) = \min_{-1 \le i,j \le 1} f(x+i, y+j)$$

placing each result in memory location 1 of the corresponding processor in the array.

In this program the maximum value of each window of 9 pixels is to be used to replace the centre pixel. The CSWAP instruction allows a bigger value to replace the existing one.

Assume that we do not want to overwrite *f(x,y)*.

```
MOVE       0,1
BROADCAST  1
INE        2
INW        3
CSWAP      1,2
CSWAP      1,3
BROADCAST  1
INN        2
INS        3
CSWAP      1,2
CSWAP      1,3
```

g(x,y) is in location 1.

(10)

NLS / NA