DEPARTMENT OF ELECTRONIC AND ELECTRICAL ENGINEERING
Autumn Semester 2009-2010
Answers to Advanced Computer Architectures Questions 1…4

1. *A processing pipeline consists of 4 functional blocks, A, B, C, and D (implementing functions $f_A(\bullet)…f_D(\bullet)$) that appear, schematically, as shown in* ***Figure 1****.*

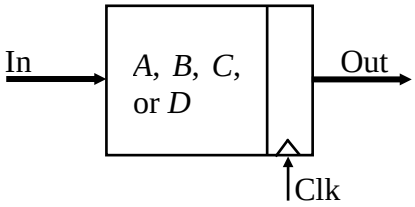
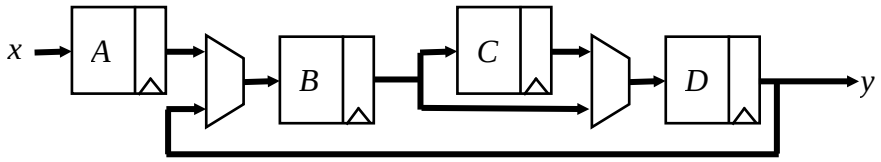


Figure 1: Functional Block

A sequence of data, x, is passed through the pipeline producing an output y such that $y_i = f_D(f_C(f_B(f_D(f_B(f_A(x_i))))))$.

a. *Draw a schematic diagram for the processing pipeline (you will need to use 2:1 multiplexers as well as only one instance of each functional block).*



(6)

b. *For the sequence of operations above:*

i) *Draw the reservation table.*

Clk	A	B	C	D
1	x_1			
2	x_2	x_1		
3		x_2		x_1
4		x_1		x_2
5		x_2	x_1	
6	x_3		x_2	x_1
7	x_4	x_3		x_2
8		x_4		x_3
9		x_3		x_4
10		x_4	x_3	
11			x_4	x_3
12				x_4

(4)

ii) *What is the throughput of the processing pipeline (in datum/clock cycle)?*

2/5 datum per clock cycle. **(2)**

iii) *What is the latency of the processing pipeline (in clock cycles)?*

6 clock cycles **(2)**

c. *You recognise that adding a single register can improve the throughput of the processing pipeline.*

i) *Show how your schematic can be modified, in this way, to improve the throughput.*

The problem lies with the gap between use and reuse of B (1 clock cycle) and D (2 clock cycles). The way to improve performance is to harmonise this gap. Looking at the reservation table/schematic, the easiest way is add a pipeline register in the bus that bypasses B to the input of D. **(3)**

ii) *Draw the reservation table for the revised schematic.*

The effect on the reservation table is:

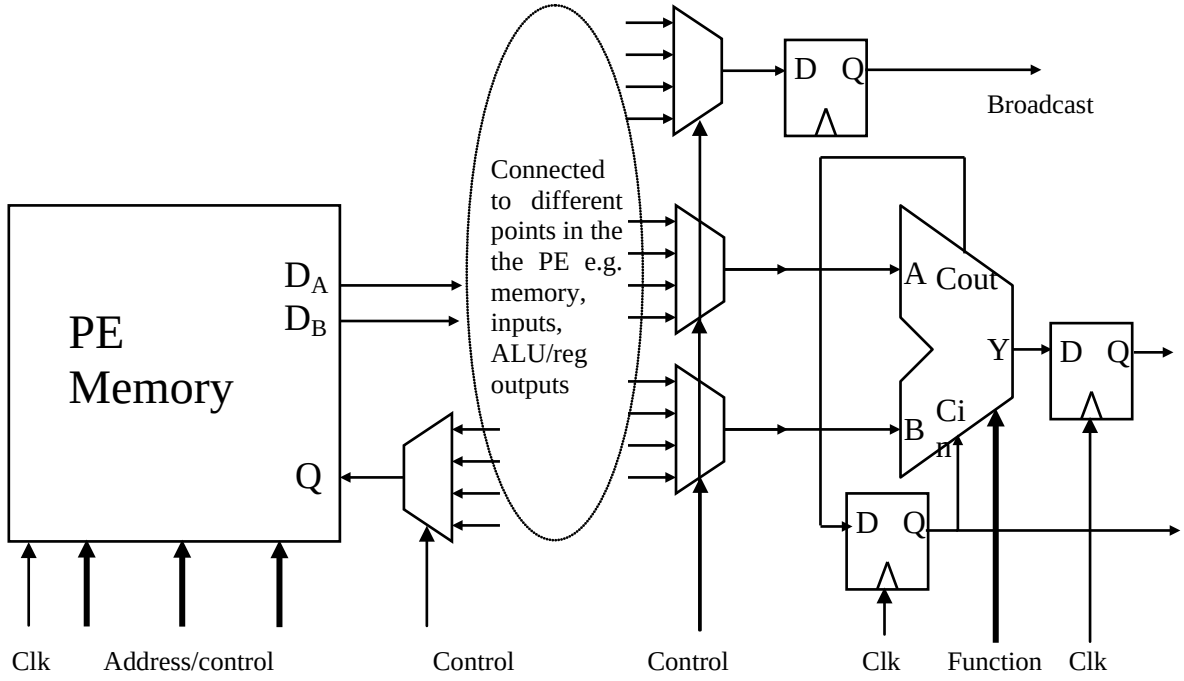
Clk	A	B	Rg	C	D
1	x_1				
2	x_2	x_1			
3	x_3	x_2	x_1		
4		x_3	x_2		x_1
5		x_1	x_3		x_2
6		x_2		x_1	x_3
7	x_4	x_3		x_2	x_1
8	x_5	x_4		x_3	x_2
9	x_6	x_5	x_4		x_3
10		x_6	x_5		x_4
11		x_4	x_6		x_5
12		x_5		x_4	x_6
		x_6		x_5	x_4

(2)

iii) *What is the improved throughput of the processing pipeline (in datum/clock cycle)?*

The relatively inexpensive change has improved the throughput from 0.4 datum/clock to 3/6 = 0.5 datum/clock – a 25% improvement in performance. **(1)**

2. a. i) *Draw a schematic diagram for the Processing Element (PE) of a typical SIMD array processor.* **(4)**



ii) *Comment on the organisation of the local storage that the PE accesses (i.e. single/dual port memory, registers) and the effect that this organisation can have on performance.*

The main issue is access to data. With a single ported memory block (assuming one clock cycle per access), an operation c = a OP b will take 3 clock cycles if the operands are coming from memory (or 4 if you assume that the write back from the flip flop at the output of the ALU cannot be overlapped with something else). This impacts on performance quite heavily. Ways to reduce this include partitioning the memory into more than one physical block (which can be restricting in terms of availability of memory) or increasing the number of ports (which adds overhead in area and control lines). If you can do two reads and a write in a single cycle then this gives the possibility of performing a dyadic infix in one clock cycle. However, in this case the critical path becomes a concern (i.e. memory read, operate, write back). If the clock frequency has to be limited because of this then every aspect of the PE's behaviour is impacted. By adding a register at the output of the ALU (essentially, a pipeline register) then this critical path can be cut to memory read, operate. **(4)**

b. *An nxn SIMD processor supports a flag, F, that allows conditional execution of instructions across the array and supports the following set of instructions:*

```
INx     a       ; copies the data from the x (=N,S,E,W) input into address a
BCAST   a       ; broadcast the data from address a to the N,S,E, and W outputs
CMP     a, #N   ; compares the data in address a with the constant N and sets
                ; the flag, F, to 1 if they are equal, 0 otherwise
INCC    a       ; increment the data in address a but only if F=1
CLR     a       ; clear the data in address a
ADD     a,b     ; add the data in address a to the data in address b
MOVE    a,b     ; copy the data in address a to the data in address b
```

An nxn image, f, is mapped onto the array such that f(x,y) is in memory location 0 of PE_{xy}. The image contains only 0s and 1s. Write a program that will detect the occurrence of patterns in the image as shown in ***Figure 2****:*

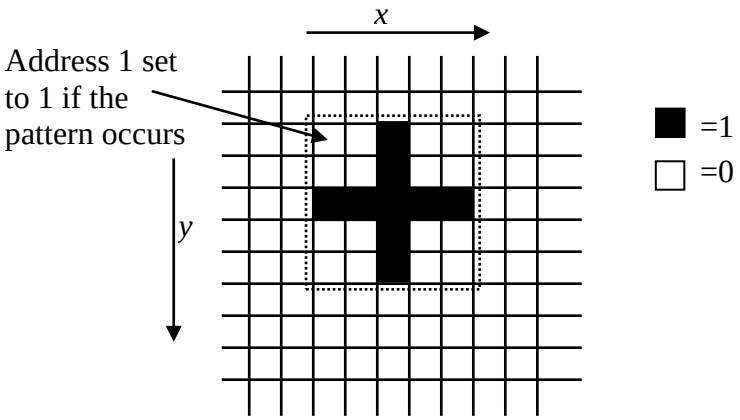


Figure 2: Pattern in Image

such that, if the exact pattern, as shown within the dotted square in ***Figure 2*** *occurs in the image, f, then the data in address 1 of the PE at the top left hand corner of the dotted square will be set to 1 otherwise it will be set to 0. Clearly, the pattern could occur anywhere in the nxn image and the program should*

detect all instances (ignoring boundary effects).

Hint: what does the pattern look like if you add up the data along each row and along each column?

By adding 5 rows up to the top then the horizontal data created is 1,1,5,1,1 for a pattern as shown (and other patterns, of course). However, by also adding 5 columns to the left then the vertical data created is also 1,1,5,1,1 and the combination uniquely identifies the pattern:

```
CLR     2       ; Clear address 2 – we will use it as a counter
MOVE    0,1     ; make a copy of the data
BCAST   1       ; broadcast                                          ┐
INS     1       ; copy it in from the south side                     │ 5
ADD     0,1     ; add the image value to the incoming value          ┘ times
CMP     1, #1   ; compare the result with one                        ┐
INCC    2       ; increment the counter if it is correct             │ 2
BCAST   1       ; copy the result to the west                        ┘ times
INE     1       ;
CMP     1, #5   ; compare the result with 5
INCC    2       ; increment the counter if it is correct
BCAST   1       ; copy the result to the west
INE     1       ;
CMP     1, #1   ; compare the result with one
INCC    2       ; increment the counter if it is correct
BCAST   1       ; copy the result to the west
INE     1       ;
CMP     1, #1   ; compare the result with one
INCC    2       ; increment the counter if it is correct
```

at this point address 2 will have the value 5 in it if the horizontal pattern is correct. For the vertical data, repeat the code above (without the initial CLR) with the INS replaced by INE and the INE replaced by INS. At the end of the second phase, address 2 will have 10 in it if the pattern is correct and so, the final instructions below complete the program:

```
CLR     1       ; Clear address 1 – this is where the result will be
CMP     2, #10  ; compare the counter with 10
INCC    1       ; increment the result if the counter is correct
```

(12)

3. a. *State Flynn's Taxonomy and give an example for each part of the taxonomy.*

Single Instruction Single Data (SISD): This describes a simple uni-processor where a single program is executed on one set of data.

Single Instruction Multiple Data (SIMD): This describes, for example, an set of processors which are constrained to execute the same program one separate sets of data. The implication being that the overall task will be executed more quickly because each part is executed in parallel. This paradigm requires that the task is regular and can be decomposed.

Multiple Instruction Multiple Data (MIMD): This describes a general purpose multi-processor where multiple processors can each work independently on their own data. The uselessness of the classification is obvious because it does not describe how well the processors can co-operate in completing a set of inter-related tasks.

Multiple Instruction Single Data (MISD): This classification appears to be anomalous because it seems to imply that multiple, different operations are applied to the same data *which remains the same data*. However, this category could cover fault-tolerant processing (e.g. triple modular redundancy) where multiple processors process the same data yielding the same results. If the results from one processor differ it is voted down (the assumption being that it is in error), and is re-started with the correct data. If a processor is persistently wrong then it is disabled. The triple modular redundancy scheme requires three processors which have been designed and coded *independently* to the same specification - this avoids processors making the same mistake and implies multiple instruction streams. However, if a processor is disabled then only two processors remain making it difficult to decide which is right if a subsequent mistake occurs in either of the remaining processors. **(4)**

b. *Amdahl's Rule is:*

$$speedup = \frac{1}{1-\alpha+\frac{\alpha}{N}}$$

i) *Explain what this means, identifying the meaning of the terms.*

This rule describes the effect of parallelising an task where a proportion, α, can be parallelised N ways and the remaining proportion, 1-α, cannot be parallelised. Speedup is the effective improvement in performance over the case when no parallelism is used. Clearly, if α is small then the effect of parallelising is limited and may not pay back the cost. **(2)**

ii) *Show how Amdahl's Rule can be extended to deal with situations where the use of parallelism is more complicated.*

A generalisation of Amdahl's Rule would be where:

$(1-\alpha)+\frac{\alpha}{N}$ would be replaced by $\sum_{i \in \text{ all steps}} \frac{\alpha_i}{N_i}$ (note: $1 = \sum_{i \in \text{ all steps}} \alpha_i$)

where α_i represents an elemental proportion of the task and N_i is the degree of parallelism associated with that part of the task. **(2)**

c. *A processor farm consists of N+1 identical processors networked together. One of the processors (the master) receives blocks of data from outside the system, each of which must be processed equivalently. The processor takes an incoming block of data, once it has been input, and splits it between the N processors (workers) actually doing the work. When a worker processor has finished processing its part of the data block, it sends the result to a master processor that is responsible for assembling and outputting the results before getting the next input block.*

Tests show that the time taken to:

- *Receive a block of data at the input is 100μs*
- *Distribute a part of the block of data to or from a worker takes negligible time*
- *Process one part of the block of data in a worker is 1500/Nμs*
- *Output a result for each block is 30μs*

Restate this problem as in Amdahl's Rule, stating any assumptions that you make.

One processor is receiving a block (which takes 100μs) and is then dispatching parts of it onwards. Data, once farmed out can be processed in parallel and so, on average, this accounts for 1500/*N*μs. During this part of the task, parallelism is *N*. Once the results are collected it takes 30μs to output them and again, this part of the task cannot be parallelised. With only one processor the total time to process a block would be 100+1500+30=1630μs. The proportion of the task that can be parallelised is 1500/1630 = 0.92 and with *N* processors participating equally this is the degree of parallelism. Consequently:

$$speedup = \frac{1}{0.08 + \frac{0.92}{N}}$$

(4)

Estimate the best value for N if:

i) *6000 blocks per second have to be processed*

If processing using only one processor takes 1630μs then this equates to 613.5 blocks per second. To achieve 6000 blocks per second then speedup = 6000/613.5 = 9.78. This equates to *N*=41.34. Thus, 42 worker processors must be used if 6000 blocks per second must be processed.

(2)

ii) cost/benefit is the metric used.

In this case cost is taken into account and the cost of the system = *N*+1 and the benefit is speedup. Consequently:

$$metric = (N+1)\left(0.08 + \frac{0.92}{N}\right) = 0.08N + 1 + \frac{0.92}{N}$$

$$\frac{dmetric}{dN} = 0 = 0.08 - \frac{0.92}{N^2}$$

$$N = \sqrt{\frac{0.92}{0.08}} = 3.4$$

(2)

On balance, N=4 is better because *metric*=1.55 as opposed to 1.62 for 3.

How might you change the system's architecture to improve performance?

The bottleneck is the serial processing on the master processor that limits the speedup to 12.5 regardless of the value of N. However, if some of the processors were diverted to being masters, overlapping the input and output of data, then the performance could be improved significantly. Alternatively, if the system was changed to allow the overlap of inputting data with processing by the workers then this would result is a balanced system when the time taken to process the data by the workers was equal to and overlapped with the time taken to input/output the data. **(4)**

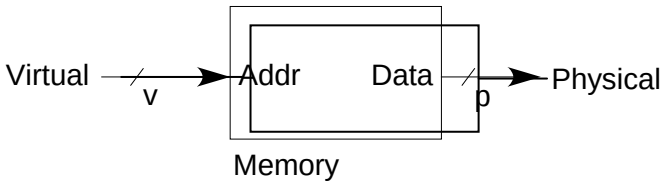
4. a. *Describe the organisation of typical memory hierarchy for a leading-edge desk-top processor system giving the general characteristics of each part of the hierarchy.*

Name	**Technology**	**Size**	t_{lat}	t_{tr}	B
Primary Cache	On-chip	8-32Kb	-	0.3ns	16-64 bytes
Secondary Cache	On-chip	64-512Kb	0.6ns	0.3ns	64-256 bytes
Tertiary Cache	On-chip	1-2Mb	1.2ns	0.6ns	0.25-1Kb
Main Memory	SDRAM	0.25-4Gb	30ns	1ns	1-4Kb
Secondary Memory	Disk	1-100Tb	1ms	30-90ns	-

(4)

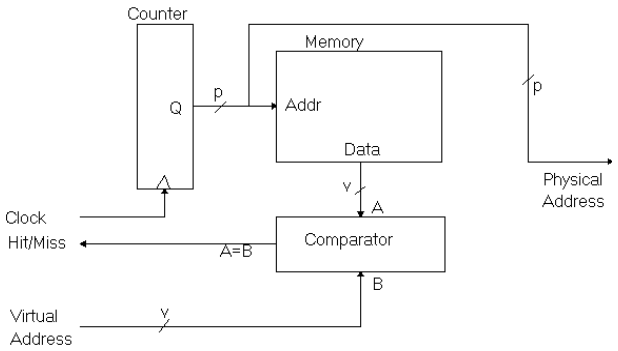
b. i) *What are the problems/issues involved in constructing a mapping scheme to map virtual to physical memory addresses.*

The internal operation of the translator is an important issue. Consider the mapping between main memory and the cache (primary or secondary) as shown below:



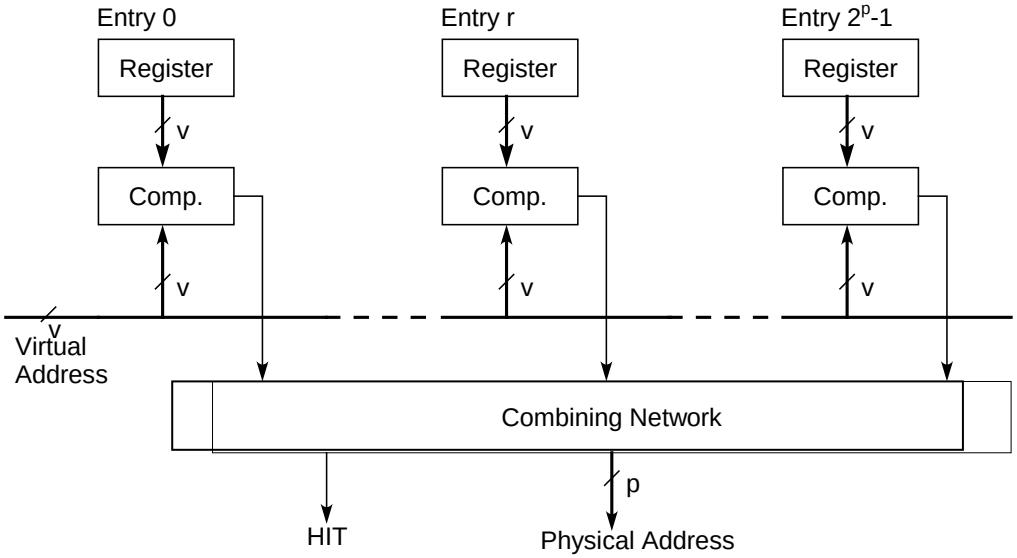
A simple look-up-table (LUT) may be used (as shown above), but if virtual address space is 2^{40} bytes (1TB), and the physical address space is 2^{32} bytes (4GB) for example, and the size of each block mapped into main memory is 2^{12} bytes (4096 bytes) then the LUT would require 2^{28} entries - 256M 20 bit words. This size would prohibit its use.

The operation of the LUT could be inverted. A normal LUT is implemented as a memory device where address:data holds the v:p relationship. However, consider the inverse scheme where address:data holds the p:v relationship:



In this case the memory device would be much smaller with a 20 bit address 1M x 28 bits. However, it would now need to be searched sequentially and this would be prohibitively slow.

A parallel solution (TLB) is required and a schematic for this is shown in below:



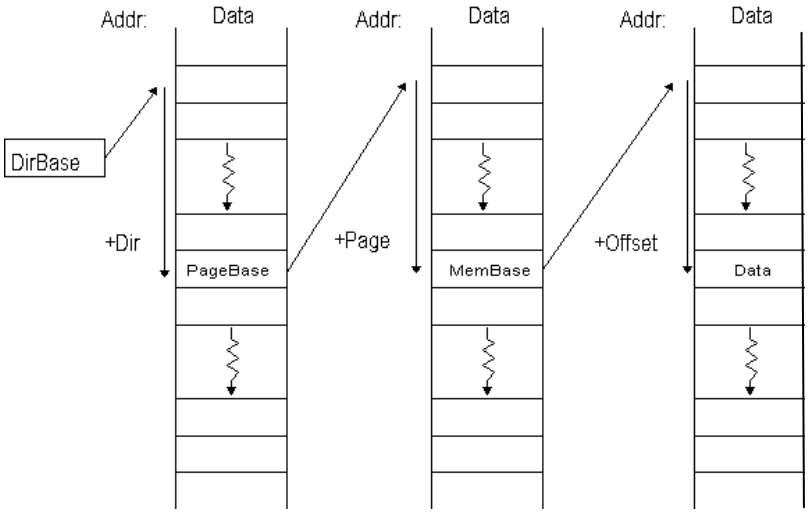
The incoming value is compared in parallel with each entry yielding a hit/miss value for each entry simultaneously. The combining network yields a corresponding value, and hit/miss output. Unfortunately, an inverse LUT of this type would occupy a lot of silicon and would be impractical it if were the primary translation mechanism. However, it would be fast. **(5)**

ii) *Describe a practical mapping scheme to help overcome these problems.*

A more practical method may rely on a combination of methods. Consider the virtual address applied to the translator is split up into 3 fields:

DIR	PAGE	OFFSET

Associated with this is a register, DirBase, which identifes the base address of a table which is held in main memory. When an address is issued, the following look-up process takes place as shown in the diagram.

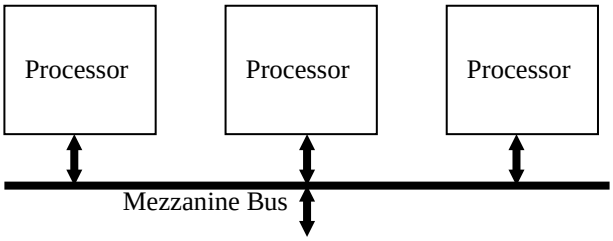


The process appears to be cumbersome and slow but it has some advantages. The tables are sparse (i.e. only those describing pages in the memory need to be **(3)**

there). However, this method is slow, requiring three accesses into main memory, and so this method can be augmented by adding a TLB to the translator. A successful look-up will yield the data but it will also yield the relationship, dir:page → MemBase. This relationship can be loaded into the TLB which holds the *m* most recently accessed blocks (where *m* is quite small). Thus, when an access takes place, the TLB is checked and yields the relationship, dir:page → MemBase much more quickly than the look-up can. If the relationship cannot be found then the look-up process is used and the relationship is then loaded into the TLB. If the look-up process cannot find the relationship then the data is not mapped into main memory.

c. *Increasingly, leading-edge systems consist of multiple processing cores that share a common memory system. Describe the problems that this gives rise to and how these problems can be resolved (your answer should cover coherency, contention, and bandwidth).*

The typical architecture of such a system might be:



Clearly, the exact organisation of the bus and connections depends on the cache. A typical arrangement might be for each processor core to have its own L1 and L2 caches and then share a much bigger L3 cache. Clearly, this puts significant demands on the L3 cache in terms of keeping the processors supplied with data, although this is offset by the L1/2 caches. Maintaining coherency across data being accessed by multiple processors is dealt with in a number of ways. At the lowest level, dynamic coherency schemes and/or snooping schemes are used to track ownership of cache lines and invalidate/flush data in caches when a potential coherency problem is detected allowing an orderly change in ownership. This is more simple to do quickly because the cores are on the same IC. Furthermore, at a higher level, semaphores/locking can be used to enforce orderly access to most shared data in accordance with the needs of the applications – cutting down the instances when processors actually vie for data. In terms of memory bandwidth, increasingly, multiple core processors have a number of parallel memory connections externally directly from the IC to provide high rates of transfer and separate connections to other parts of the memory sub-system (e.g. disk). Reordering buffers within the cores can group transfers together to minimise the amount of traffic between the cores' local caches and the L3 cache. Finally, at the interface between the IC and memory, techniques such as split transaction buses can be used to make the best of the available bandwidth. **(8)**

NLS