UNIVERSITY OF VICTORIA

EXAMINATIONS FALL 2005

C SC 230 - Introduction to Computer Architecture and Assembly Language

STUDENT NUMBER: _____

TIME: 3 hours

INSTRUCTORS: M. Serra and R. N. Horspool

TOTAL MARKS: 112

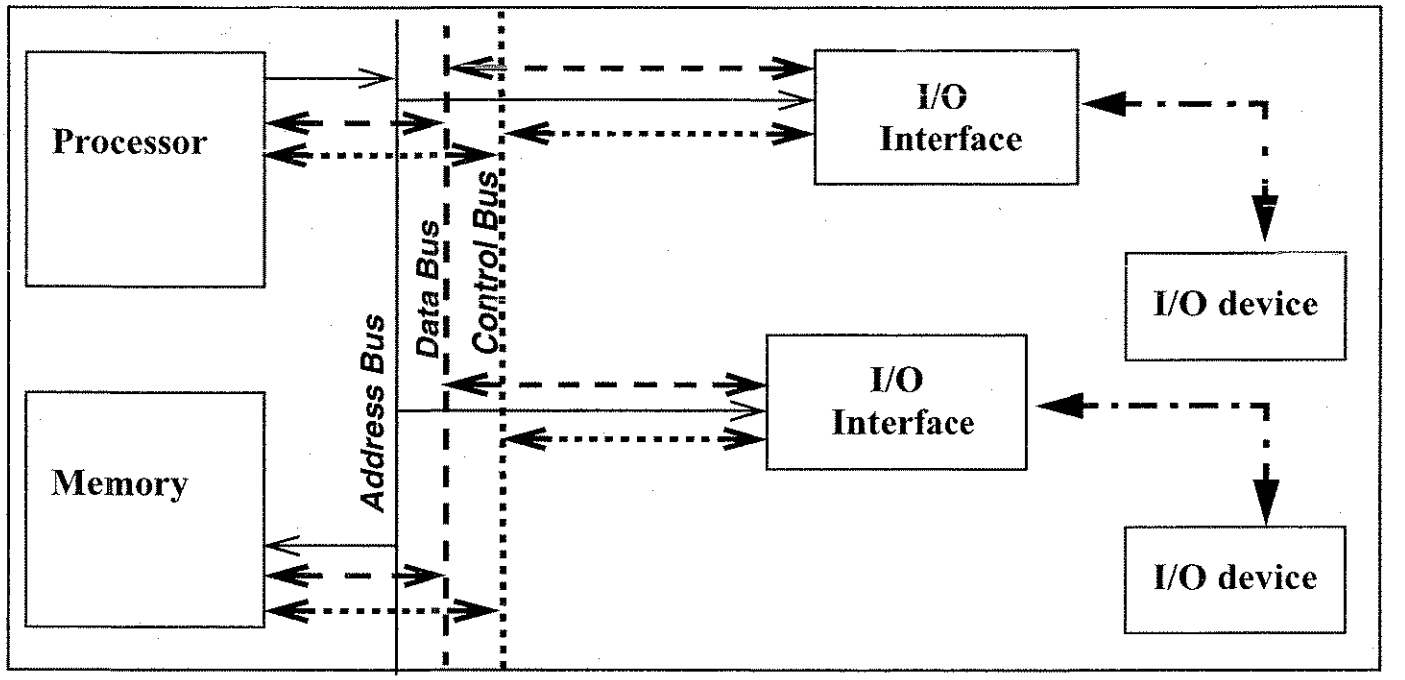
TO BE ANSWERED ON THE PAPER

Question No.	Value	Mark	Question No.	Value	Mark
1	*8*		7	*12*	
2	*10*		8	*15*	
3	*10*		9	*12*	
4	*5*		10	*12*	
5	*13*		11	*6*	
6	*9*		TOTAL	112	

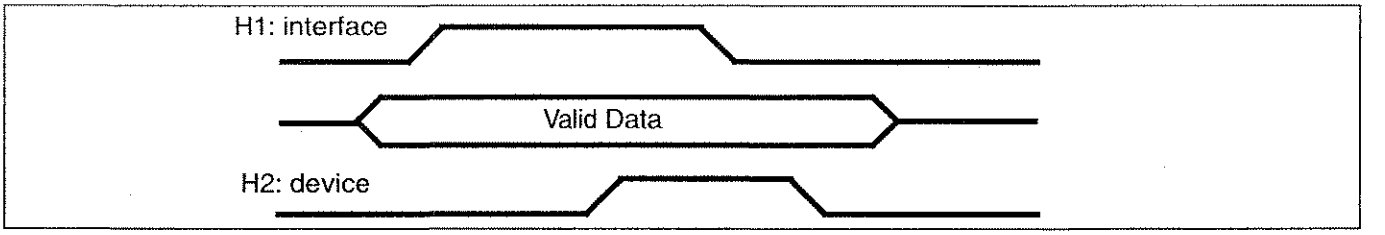
INSTRUCTIONS:

1. STUDENTS MUST COUNT THE NUMBER OF PAGES IN THIS EXAMINATION PAPER BEFORE BEGINNING TO WRITE, AND REPORT ANY DISCREPANCY IMMEDIATELY TO THE INVIGILATOR
2. This examination paper consists of 14 pages including this cover page.
3. No aids are permitted. However, a handout describing the ARM instruction set is provided for your use.
4. The marks assigned to each question and to each part of a question are shown within square brackets. Partial marks are available for all questions
5. Please be precise but brief, and use point form where appropriate.
6. It is strongly recommended that you read the entire exam through from beginning to end before beginning to answer the questions.

Question 1. [8] You have seen this diagram before, showing some possible interconnections between peripheral devices and a processor using interfaces. In the second test, you gave a brief description of the possible schemes of communications often applied between either the processor and interfaces (polling or interrupts) or between the interfaces and the devices (strobes or a full handshake protocol).



(a) [5] Consider the timing diagram below describing the timing events for an *output handshaking protocol* using *interlocked mode,* as discussed in the lectures and explain precisely where it applies, what happens at every step, why, what the signals mean and anything else, briefly, to show your understanding.

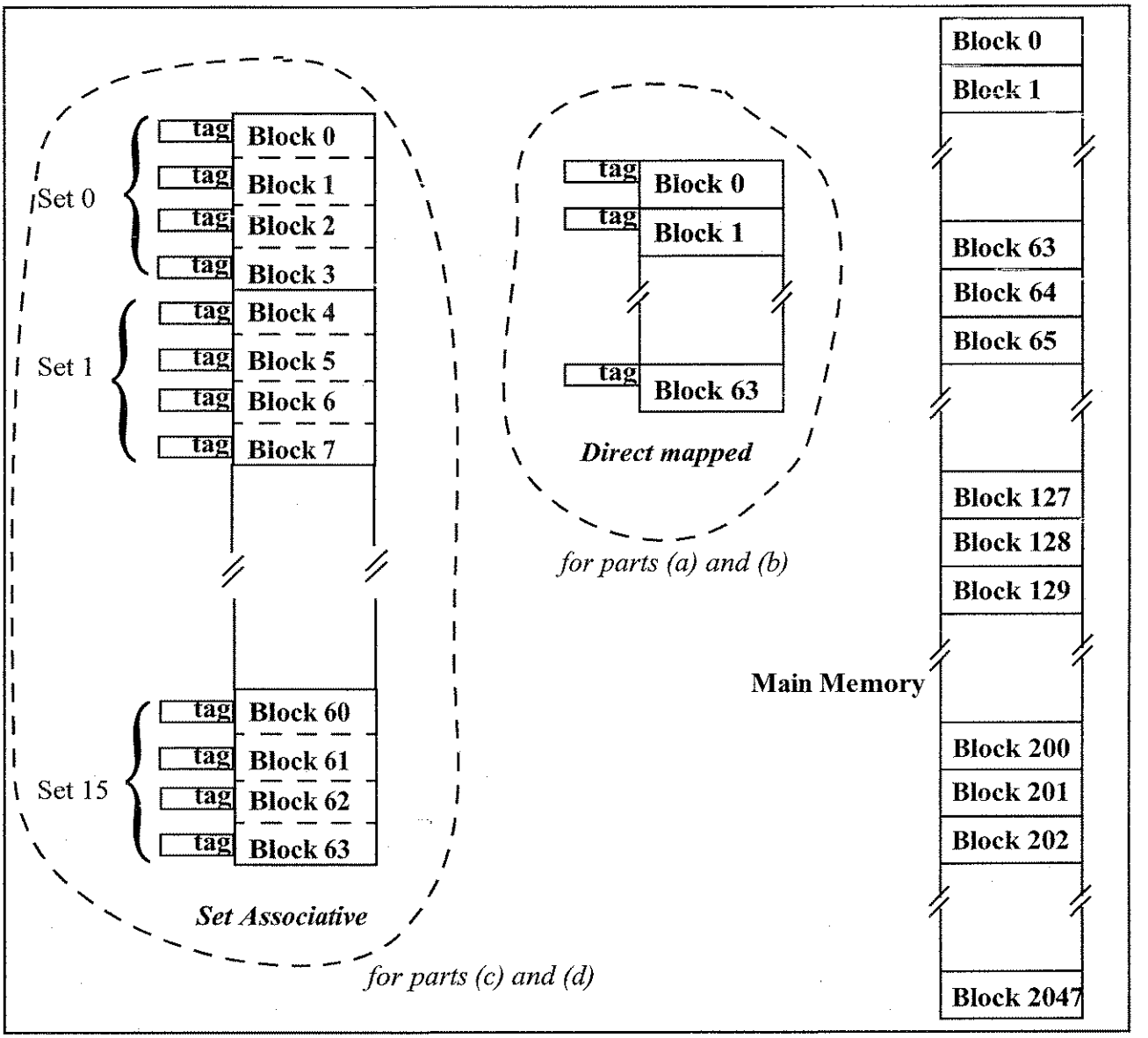


(b) [3] Explain "polling" as a synchronization method for communication between a processor and peripherals through their interfaces.

Question 2. [10] For each statement in the list below, place a **T** or **F** mark in the right-hand column to indicate whether it is true (T) or false (F).

Big-endian architectures have the sign bit in the byte with the biggest address	
In two's complement representation, there are fewer positive numbers than negative numbers	
In two's complement representation, the result of performing an arithmetic shift right by one place on the value -1 leaves that value unchanged	
In floating point arithmetic, it is possible that the expressions `a+b+c` and `c+b+a` give different results	
Each device attached to the computer requires a separate interrupt routine	
Every instruction must occupy the same number of bytes of memory	
A data hazard occurs if the result of an arithmetic operation is too large to be held in the result register	
A page fault interrupt occurs if a parity error is detected while reading a page from disk	
An interrupt routine on the ARM processor can itself be interrupted	
A subroutine must always return control to its caller	

Question 3. [10] [Consider a main memory of 32K words (= 2^{15}words), addressable by 16-bit addresses, to be viewed as 2K blocks (= 2^{11} blocks) of 16 words each, (2^{11} blocks x 2^4 words =2^{15}words), as shown in the figure below. Consider also a cache consisting of 64 blocks of 16 words each, for a total of 1024 (1K) words.



(a) [2] Assume that a direct mapping configuration is used for the cache organization. Describe precisely how and where a generic block *K* from memory is loaded into the cache by summarizing the general strategy and giving the block number in the cache which corresponds to block *K*. The figure above shows the blocks in the direct mapped cache in the centre.

(b) [2] Show that you understand the strategy by giving the cache block numbers assigned to the following *sequence* of blocks fetched from memory – the blocks are fetched in the order as written from left to right (there might be collisions).

Memory Block #	**0**	**1**	**65**	**200**	**202**	**128**	**265**
Cache Block #							

(c) [2] Assume now that a set associative mapping organization is used for the same cache, seen as 16 sets of 4 blocks each. Describe precisely how the mapping is configured and state how and where a generic block *K* from memory is loaded into the cache. The figure above also shows the blocks and set numbers for the set associative mapped cache on the left.

(d) [4] Repeat part (b) assuming set associative mapping for the same cache, seen as 16 sets of 4 blocks each. Give the cache set and block numbers assigned to the following *sequence* of blocks fetched from memory – the blocks are fetched in the order as written from left to right (there might be collisions).

Memory Block #	**0**	**1**	**65**	**200**	**202**	**128**	**265**
Cache Set #							
Cache Block #							

Question 4. [5] You are given the following scenario:

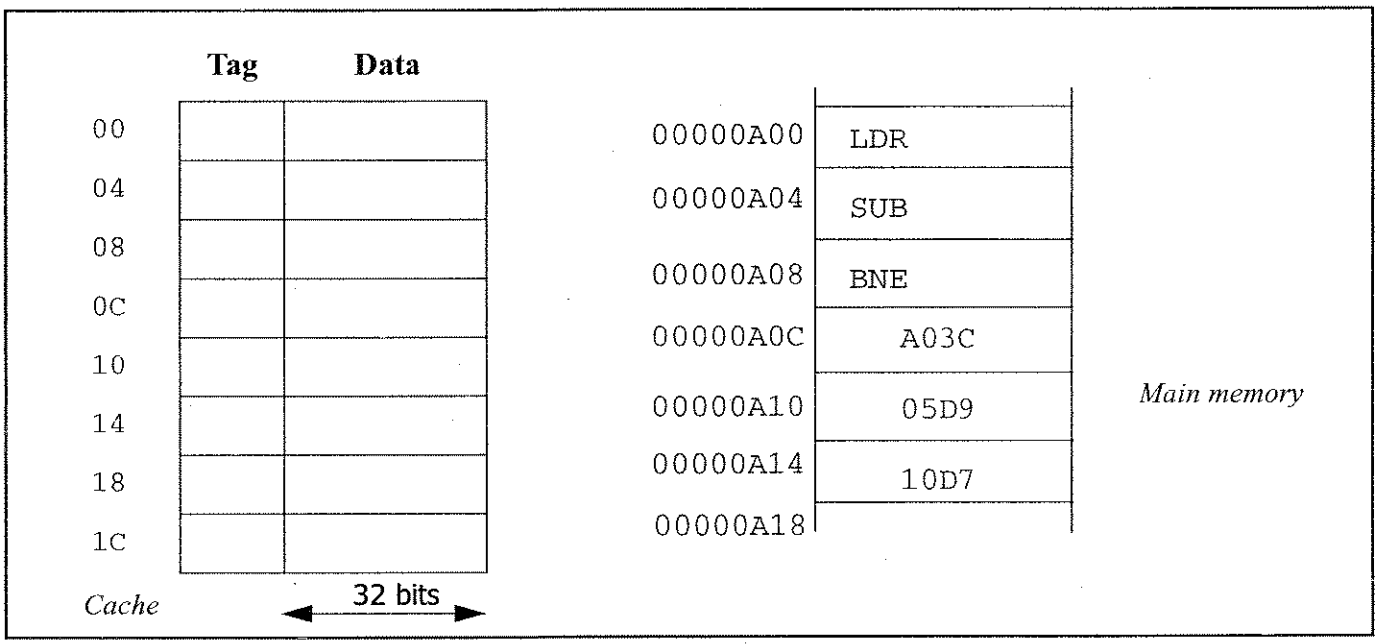
- a processor has on-chip cache and memory accessible through the system bus;
- execution time for a program is directly proportional to instruction access time;
- access to an instruction in the cache is 50 times faster than access to an instruction in the main memory;
- a requested instruction is found in the cache with probability 0.96;
- if an instruction is not found in the cache, it must first be fetched from main memory to the cache and then fetched from the cache to be executed – thus the total is 1 memory access + 1 cache access.

Compute the ratio of program execution time without the cache to execution time with the cache – this ratio is usually defined as the speed-up factor resulting from the presence of the cache.

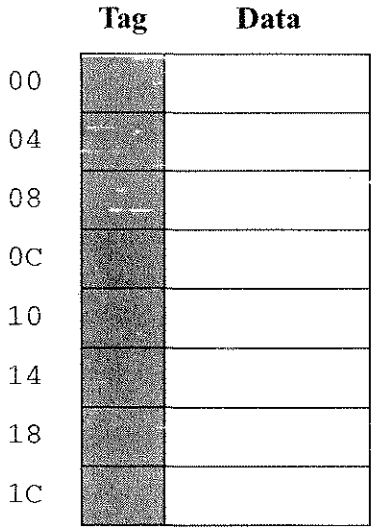
Question 5. [13] A computer uses a direct-mapped cache between the main memory and the processor. It contains eight 32-bit words, each with a tag. When a miss occurs during a read operation, the requested word is read from memory and sent both to the processor and copied into the cache. The rightmost 5 bits of the memory addresses are used to compute the cache block numbers. Consider the following ARM code:

```
00000A00   [1]   HERE:LDR   R1,[R2,#4]!
00000A04   [2]         SUB   R3,R3,#1
00000A08   [3]         BNE   HERE
```

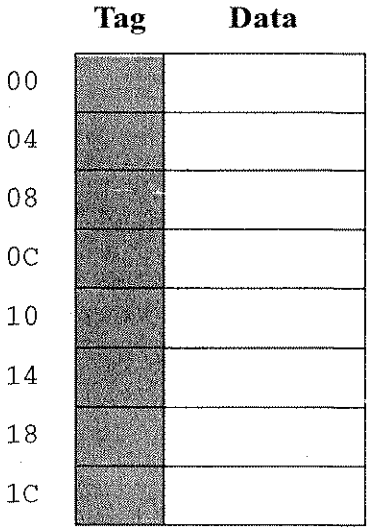
The numbers on the left side are the addresses where the instructions have been loaded in memory, while the second column of numbers in square brackets denotes a line number. Before execution, registers R1, R2, and R3 contain `0x0000`, `0x0A08`, and `0x0003`, respectively. The contents of the relevant portion of memory with addresses are shown on the right hand side of the figure below, with all entries in hexadecimal notation.



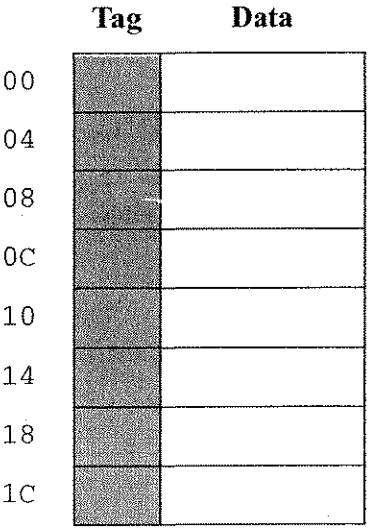
(a) [9] Show the contents of the cache after each instruction. To help you (and to help marking), there are many drawings of the cache for you to fill. You should leave the tag parts blank.



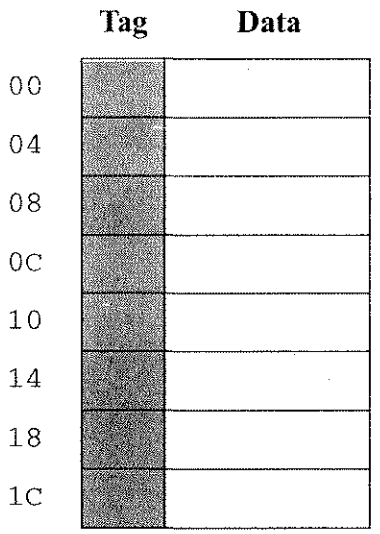
Iteration 2, after line [1]



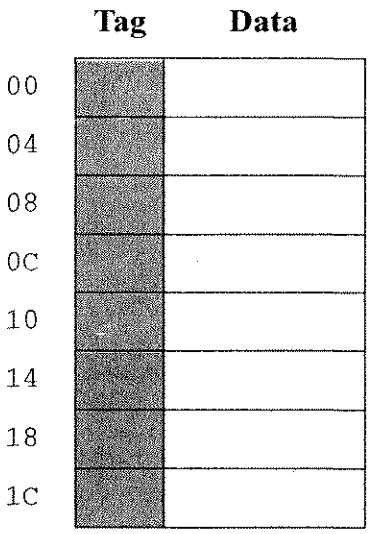
Iteration 2, after line [2]



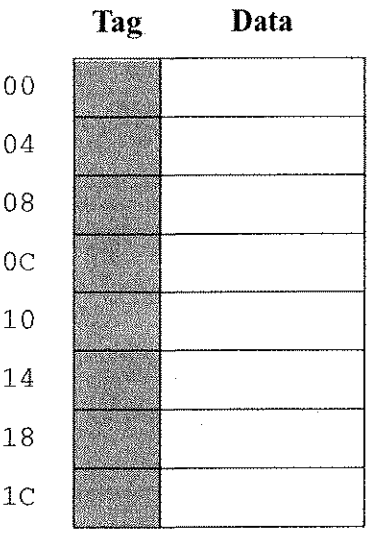
Iteration 2, after line [3]



Iteration 3, after line [1]



Iteration 3, after line [2]



Iteration 3, after line [3]

(b) [2] Assume that the access time of the main memory is *20t* and that of the cache is *2t*. Calculate the execution time for each pass. Ignore the time taken by the processor between memory cycles.

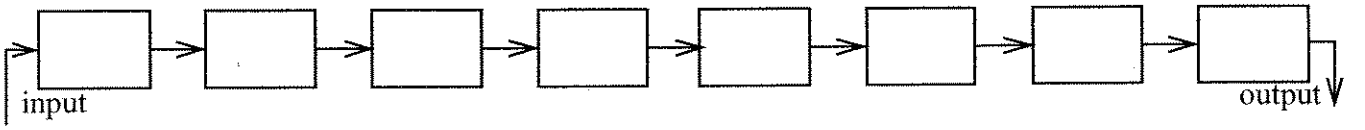
(c) [2] If the cache had been divided into instruction and data caches, do you think that the access time would be lower in this case? What can you say about the general case? (Answer both parts.)

Question 6. [9]

(a) [3] When describing the functionality of a pipelined execution processor, three possibilities for hazards, which can reduce the performance increase, were described. State what hazards are possible and give their definitions, perhaps using an example as well for clarity.

(b) [2] Suppose that you need to execute 152 processes and that each process requires 12 clock cycles. In total, the 152 processes would take $152 \times 12 = 1{,}824$ clock cycles.

Now suppose that you are given a pipeline with 8 stages to perform the processes, as drawn below.



How many clock cycles will it take if there are no hazards? Show the answer as a number and also show how you calculated it.

(c) [2] When evaluating the performance of a pipeline, we calculate the possible speedup as the ratio:

$$Speedup = \frac{T_{serial}}{T_{pipeline}}$$

What is the speedup that can be obtained using the pipeline of part (b)?

(d) [2] In general, the speedup can be calculated as:

$$Speedup = \frac{T_{serial}}{T_{pipeline}} = \frac{m \times N}{m + N - 1}$$

where m denotes the number of stages in a pipeline and N denotes the number of items to be processed. If one assumes that $N \gg m$, what can you say about the asymptotic speedup which could be obtained in theory?

Question 7. [12] The figure below, copied from your textbook, shows a possible view of Virtual Memory. Yet the label "Virtual Memory" does not explicitly appear anywhere.

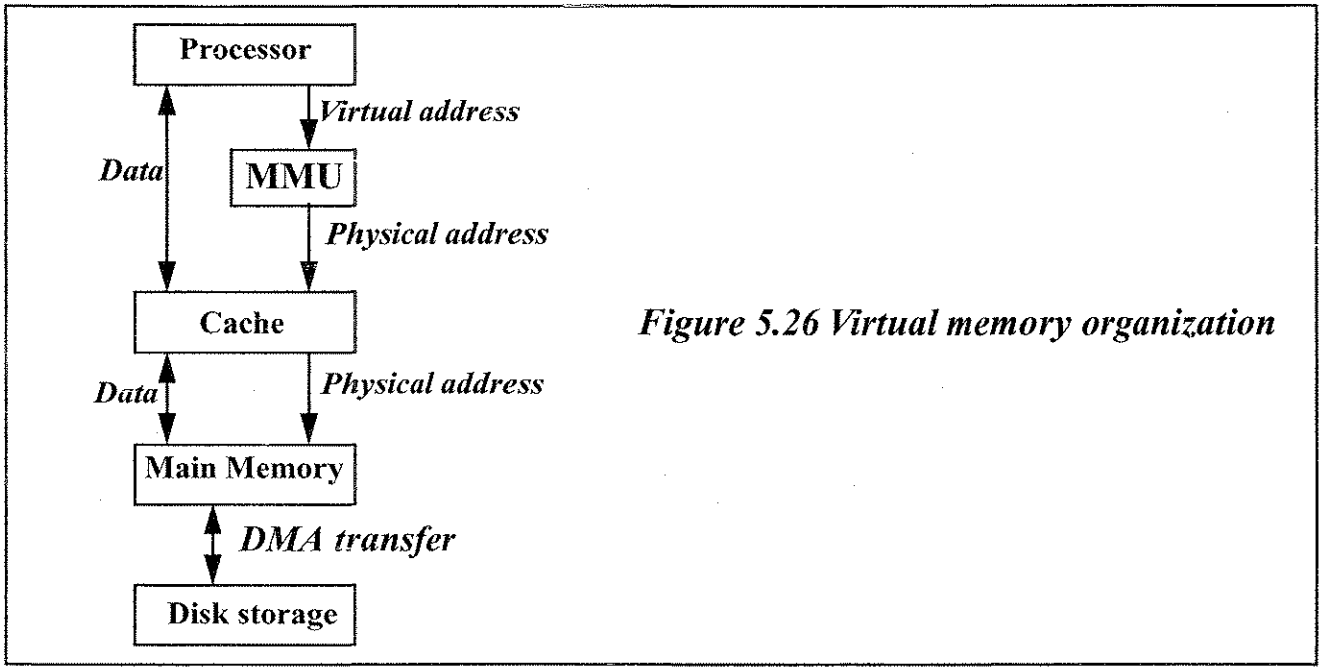


Figure 5.26 Virtual memory organization

(a) [2] Give a brief definition of Virtual Memory.

(b) [2] What benefits does Virtual Memory provide?

(c) [2] What is the function of the MMU?

(d) [2] What is the function of DMA?

(e) [2] Is the presence of DMA essential for the implementation of Virtual Memory? Why or why not?

(f) [2] In a system design, should Virtual Memory be considered a component of hardware, software or both? Why?

Question 8. [15] The subroutine *Identity* is required to initialize the elements of a $N \times N$ matrix to be all zeroes, except for the elements along the diagonal which are initialized to one. For example, a 3×3 matrix should be initialized to the values shown on the right.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(a) [5] Provide high-level pseudocode for the *Identity* subroutine. It is called with two parameters – a $N \times N$ matrix of integers and the value of N.

(b) [10] Provide ARM code for the *Identity* subroutine which matches, as closely as possible, your pseudocode given as the answer to part (a). The address of the matrix is passed in `R1` and N is passed in `R2`. No result is returned in any register. The matrix elements are arranged consecutively in memory so that an element A[i, j] has the address A + ($i \times N + j$)×4, where $0 \le i < N$, $0 \le j < N$.

```
; Identity(MatrixAddress, N)
;     initializes a N by N matrix at address MatrixAddress to be
;     an identity matrix. On entry, R1 = MatrixAddress, R2 = N.
Identity:
     stmfd sp!,{r0-r10,lr}   ; save all registers

     ldmfd sp!,{r0-r10,pc}   ; restore registers and return
```

Question 9. [12] The C language provides two different ways to process the elements of an array, as illustrated by the following two versions of a function for summing the elements of an array.

```
int Sum1( int *A, int N ) {
      int i, s;
      i = s = 0;
      while( i<N ) {
            s += A[i]; i++;
      }
      return s;
}
```

```
int Sum2( int *A, int N ) {
      int *AEnd, s;
      s = 0;
      AEnd = A+N;
      while( A < AEnd ) {
            s += *A; A++;
      }
      return s;
}
```

(a) [2] What is the C datatype of A in *Sum1* and *Sum2*? Tick the most appropriate answer below:

☐ **int** ☐ **array of int** ☐ **pointer to int** ☐ **pointer to array of int**

(b) [3] Assuming that `A`, `N`, `i` and `s` are held in registers R1, R2, R3 and R4 respectively, give the ARM assembly language code which corresponds to the two statements `s+=A[i]; i++;` in *Sum1*.

(c) [3] Assuming that `A`, `N`, `AEnd` and `s` are held in registers R1, R2, R3 and R4 respectively, give the ARM assembly language code which corresponds to the two statements `s+=*A; A++;` in *Sum2*.

(d) [2] Suppose that *Sum2* is called with the values `0xA000` for `A` and `7` for `N`. What value, in hexadecimal, is stored in variable `AEnd` by the statement which initializes it? Tick the correct answer.

☐ `0xA01C` ☐ `0xA028` ☐ `0xA018` ☐ `0aA007`

(e) [2] An experienced C programmer would probably write the single statement `s+=*A++;` instead of the two statements `s+=*A; A++;` in *Sum2*. Explain what the expression `*A++` means.

Question 10. [12] For each description or phrase listed in the second table, write the number of the item from the first table which is the closest match. (Each item in the first table can be used only once.)

1	Register
2	Cache
3	Dynamic RAM (DRAM)
4	Synchronous DRAM (SDRAM)
5	Interrupt mask
6	Pipeline

7	Condition codes
8	Handshake
9	Strobe
10	Multiplexer
11	Bus
12	Translation Lookaside Buffer (TLB)

Write each number in the column on the right →

A series of partially executed instructions in the CPU	
A group of lines which connects several devices	
A small high speed memory holding copies of memory blocks	
A cache of recent translations performed by the memory management unit (MMU)	
Capture the values on a series of data lines and store them in a buffer	
A high speed storage element holding one word of data	
A protocol for controlling data transfers on the bus	
Memory whose operation is controlled with a clock signal	
The results of a recently executed operation, commonly kept in the N, Z, V and C bits	
A circuit which selects one of its many output lines to hold the value 1	
Memory which does not retain its state indefinitely without being refreshed	
A set of bits which enable/disable interrupts	

Question 11. [6]

(a) Given the integers $+12_{10}$ and -12_{10}, give their 2's complement 4-bit binary representation.	$+12_{10}$=____$_2$	-12_{10}=____$_2$
(b) Assuming a 2's complement 8-bit binary representation, give the decimal equivalent of $5A_{16}$ and $A5_{16}$	$5A_{16}$=____ 10	$A5_{16}$=____ 10
(c) Give the hexadecimal equivalent of the 8-bit binary number 10100101.	10100101_2 = ____ 16	
(d) Given the hexadecimal number $A5_{16}$, give its decimal representation assuming it is an unsigned integer.	$A5_{16}$ = ____ 10	

END

CSC 230

CSC230 Pg2

Operand 2	
Immediate value	#<immed_8>
Logical shift left immediate	Rm, LSL #<immed_5>
Logical shift right immediate	Rm, LSR #<immed_5>
Arithmetic shift right immediate	Rm, ASR #<immed_5>
Rotate right immediate	Rm, ROR #<immed_5>
Register	[Rm]
Rotate right extended	Rm, RRX
Logical shift left register	Rm, LSL Rs
Logical shift right register	Rm, LSR Rs
Arithmetic shift right register	Rm, ASR Rs
Rotate right register	Rm, ROR Rs

Condition Field	
EQ	Equal
NE	Not equal
CS	Carry Set
CC	Carry clear
MI	Negative
PL	Positive or zero
VS	Overflow
VC	No overflow
HI	Unsigned higher
LS	Unsigned lower or same
GE	Signed greater or equal
LT	Signed less than
GT	Signed greater than
LE	Signed less than or equal
AL	Always

Dec	Bin	Hex
0	00000000	00
1	00000001	01
2	00000010	02
3	00000011	03
4	00000100	04
5	00000101	05
6	00000110	06
7	00000111	07
8	00001000	08
9	00001001	09
10	00001010	0A
11	00001011	0B
12	00001100	0C
13	00001101	0D
14	00001110	0E
15	00001111	0F

Addressing Mode 4 - Multiple Data Transfer			
Block load		**Stack pop**	
IA	Increment After	FD	Full Descending
IB	Increment Before	ED	Empty Descending
DA	Decrement After	FA	Full Ascending
DB	Decrement Before	EA	Empty Ascending
Block store		**Stack push**	
IA	Increment After	EA	Empty Ascending
IB	Increment Before	FA	Full Ascending
DA	Decrement After	ED	Empty Descending
DB	Decrement Before	FD	Full Descending

D	BIN	H	D	BIN	H	D	BIN	H	D	BIN	H
0	00000000	00	4	00000100	04	8	00001000	08	12	00001100	0C
1	00000001	01	5	00000101	05	9	00001001	09	13	00001101	0D
2	00000010	02	6	00000110	06	10	00001010	0A	14	00001110	0E
3	00000011	03	7	00000111	07	11	00001011	0B	15	00001111	0F