UNIVERSITY OF VICTORIA

EXAMINATIONS SUMMER 2009

C SC 230 - Introduction to Computer Architecture and Assembly Language - CRN# 30234

STUDENT NUMBER: ______________________________

TIME: 3 hours

INSTRUCTOR: M. Serra and R. Brown

TOTAL MARKS: 100

TO BE ANSWERED ON THE PAPER

Question No.	Value	Mark	Question No.	Value	Mark
1	*8*		9	*6*	
2	*4*		10	*6*	
3	*4*		11	*8*	
4	*2*		12	*10*	
5	*6*		13	*8*	
6	*18*		14	*8*	
7	*10*		**TOTAL**	**100**	
8	2				

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 19 pages including this cover page plus 2 pages of Appendix.
3. No aids are permitted. However, an Appendix describing the ARM instruction set is provided for your use.
4. The marks assigned to each 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] *Amdhal's law* was discussed,stated as:.

$$\text{Speedup} = \frac{1}{f + \frac{1-f}{p}}$$

Your task is to explain briefly what it is and in what context it is applied, as if you were giving a mini lecture. Answering the questions below should guide you.

(a) [4] Can you explain what this formula means and what each parameter is?

(b) [2] Can you give an example to show its impact when f=10% and the number of PEs used is 10?

(c) [2] What is the meaning of this law, or what is its impact, if one were to offer you to use an almost unlimited number of PEs to build the biggest supercomputer ever?

Question 2. [4] Figure 1 shows the possible organization of a 4-stage pipeline.

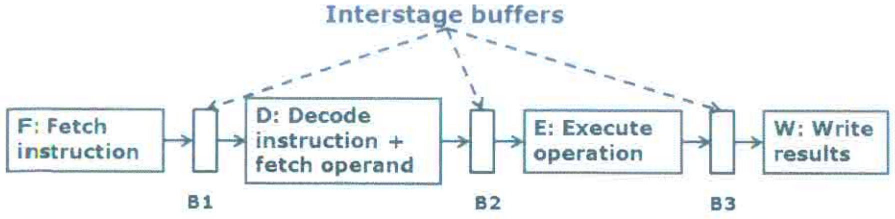


FIGURE 1. A 4-stage Pipeline

(a)[2] Redraw the diagram showing how the organization could be enhanced by including an instruction queue.

(b)[2] Redraw the diagram showing how the organization should *normally* be enhanced in a super scalar design.

Question 3. [4] When looking at parallel processing and its characteristics, three main items are considered: the number of processing units (PEs), their interconnection and the organization of memory being accessed. All three characteristics affect performance, which can be further subdivided into four evaluation points: *parallel time*, *speedup*, *efficiency* and *throughput*. Give a clear definition below using an appropriate formula for *speedup* and *efficiency.*

speedup	
efficiency	

Question 4. [2] A system requires an 8MB ROM and a 23MB RAM, all byte addressable, plus it needs another 2^{20} addresses for peripherals. What is the minimum number of lines to be used for an Address Bus? Show your work to explain your final answer [1 for answer, 1 for explanation].

Question 5. [6] Program execution time, *T*, can be defined as:

$$T = \frac{N \times S}{R} \qquad \text{where}$$

- *T* is the total elapsed time,
- *N* is the number of machine language instructions used during the execution (not necessarily the number of machine instructions in the object code),
- *S* is the number of basic steps needed to execute one machine instruction (where each basic step is assumed to take 1 clock cycle),
- *R* is the clock rate.

You are asked to examine *T* for a certain high-level language program. The program can be run on a RISC or a CISC computer. Both computers use pipelined instruction execution and have the same clock rate *R*. However pipelining in the RISC machine is more effective than in the CISC machine, such that the effective value of *Sr* for the RISC machine is 1.2, but it is *Sc* = 1.5 for the CISC machine. You are told that the program will have the same total execution time *T* on both machines if *Nc*, the number of machine language instructions for CISC, is 80% or 4/5 of *Nr*, the number of machine language instructions for RISC. Is this correct? Show your work and your reasoning in order to justify your answer in various steps, as below..

(a) [1] ***Elapsed time in RISC:*** Tr =

(b) [1] ***Elapsed time in CISC:*** Tc =

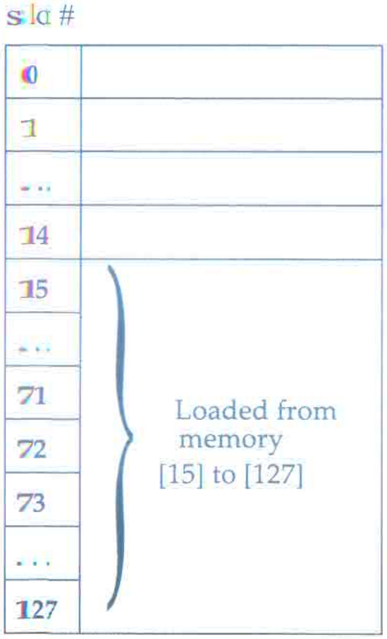
(c) [4] ***Given*** *Sc* = 1.5 ***and*** *Sr* = 1.2, ***the conjecture is that*** Nc = 4/5 Nr ***will give*** Tr = Tc. ***Why?***

Question 6. [18] A direct mapped cache consists of 128 slots of 16 words each. Main memory contains 16K blocks of 16 words each. (Reminder: 1K = 1024)

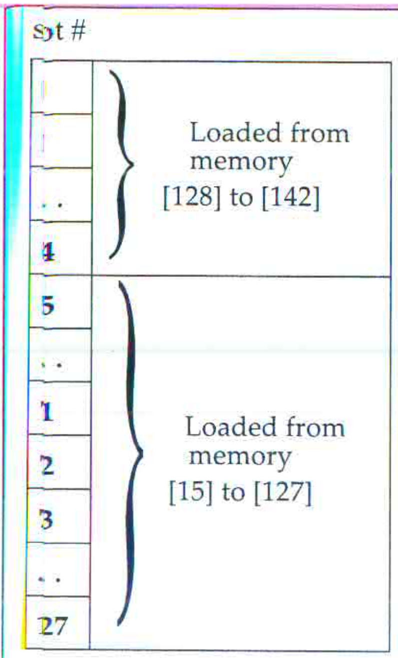
(a) [4] Draw a diagram of the organization of the cache and memory with labels. Pick at least 3 memory blocks and show where they would map into cache slots. Your 3 examples should include 2 blocks for which there is no collision and a third block for which there is a collision.

(b)[4] Draw the diagram again assuming the cache is now 2-way associative and show the hanges. Use the same examples from above to show how the memory blocks would map into he newly organized cache.

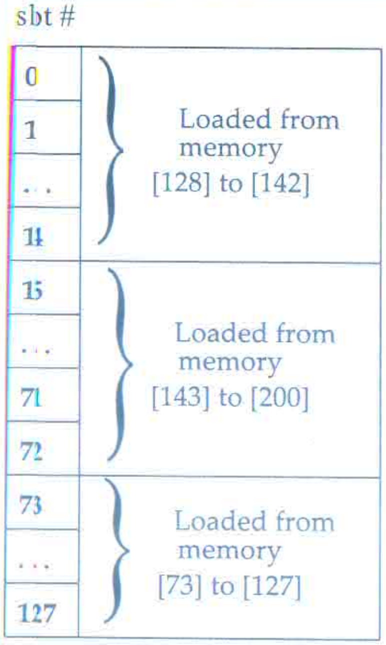
(c) ssume you have the initial direct mapped cache organization with an access time of 10 ns, and he time to fill a cache slot is 200 ns. Load-through is not used; that is, when an accessed word is ot found in the cache, the entire block is brought into the cache and the word is then accessed hrough the cache. Initially the cache is empty. You are asked to compute the hit ratio and the ffective access time for a program which loops 10 times over memory blocks 15-200. To help ou, a simulation of the first pass through the loop has been done with an explanation. Follow it hrough so that you can proceed later.



- The loop needs to execute starting at memory block [15].
- Access is requested and there is a cache miss.
- Start loading from memory block [15] to memory block [127]
- Store in cache slots (15) to (127)
- 113 accesses, all misses
- 113 @ (10 + 200) ns = 113 x 210 ns = 27,730 ns


slot #
..
4
5
..
1
2
3
..
27
Loaded from memory [128] to [142]
Loaded from memory [15] to [127]

- Continue the loop needing access and having to load from memory [128] to memory [142]
- Store in cache slots (0) to (14)
- 15 accesses, all misses
- 15 @ (10 + 200) ns = 15 x 210 ns = 3,150 ns


slot #
0
1
. . .
14
15
. . .
71
72
73
. . .
127
Loaded from memory [128] to [142]
Loaded from memory [143] to [200]
Loaded from memory [73] to [127]

- Continue loading from memory [143] to memory [200]
- Store in cache slots (15) to (72)
- 58 accesses, all misses
- 58 @ (10 + 200) ns = 58 x 210 ns = 12,180 ns

This is what the cache looks like after the first loop.

After the first loop, there have been (113+15+58) = 186 accesses to memory, all misses from the cache and all needing to load cache slots. Hit ratio = 0/186 = 0. Total time = (27,730+3,150+12,180) ns = 43,060 ns. Collisions occurred for slots (15) to (72).

[i] [6] Now repeat the simulation shown above for the second pass through the loop, accessing memory blocks [15] to [200] again. Use the empty diagrams below and explain what

happens in a similar fashion as exemplified above. At the end show the state of the cache after the second loop and compute the hit ratio for this second iteration. You will be marked for the detailed answers and explanations.

slot #

0	
1	
..	
14	
15	
..	
71	
72	
73	
..	
127	

slot #

0	
1	
...	
14	
15	
...	
71	
72	
73	
...	
127	

bt #

. .	
4	
5	
. .	
1	
2	
3	
. .	
27	

Go through the process one more time, for the third pass through the loop. The same empty diagrams are provided for you as a worksheet but this time they will not be marked. The only marks will be for answering some final questions below.

sbt #

0	
1	
. . .	
14	
15	
. . .	
71	
72	
73	
. . .	
127	

slot #

0	
1	
. . .	
14	
15	
. . .	
71	
72	
73	
. . .	
127	

slot #

0	
1	
. . .	
14	
15	
. . .	
71	
72	
73	
. . .	
127	

[ii] [4] After 3 passes through the loop you must have some idea of what is happening. Can you draw your conclusions stating what you expect to take place for the execution of this program which loops 10 times from locations 15-200? Your minimal answer will include the hit ratio for each iteration and overall, plus an explanation.

Question 7. [10] You have seen the diagram in Figure 2 before, showing some possible interconnections between peripheral devices and a processor, through the interfaces.

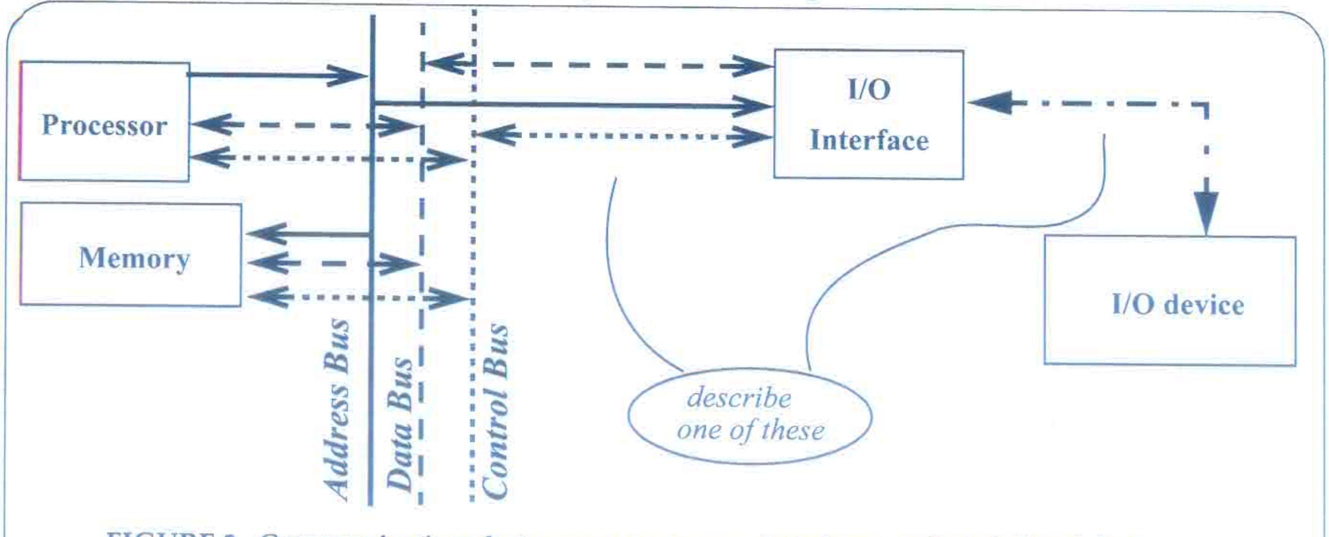


FIGURE 2. Communications between a processor, interfaces and peripheral devices.

(a) [2] State at least two possible protocols for the communication between the processor and the interfaces (no descriptions).

(b) [2] State at least two possible protocols for the communication between the interfaces and the devices (no descriptions).

(c) [] State at least one reason why interfaces are needed.

(d) [] Choose one of the protocols from (a) or from (b) and describe it briefly. You do not need to describe every detail, but show that you have clearly understood.

(e) [] Is an interface usually a software or a hardware component?

Question 8. **[2]** When running a particular program with N memory accesses, a system with a cache and paged virtual memory generates a total of M cache misses and F page faults. T1 is the time for a cache hit, T2 the time for a main memory hit, T3 the time to load a page into main memory from disk. Answer the following questions and justify all your answers to get full marks.

(a) [1] What is the cache hit ratio?

(b) [1] What is the main memory hit ratio? That is, what proportion of memory accesses do not generate a page fault?

Question 9. [6] You are being interviewed for a co-op work position. On your resume you indicate you are an experienced C and assembly language programmer. The interviewer asks you to explain the difference between static and dynamic linking of object code to library files in an Windows environment. Using good English, write down your succinct and precise response, that is, be clear and precise, but also concise (you would only be given a couple of minutes to state your answer).

Question 10. [6] Consider the high level instruction: "`A = B + C x D;`" and then consider how to translate it into a set of low level instructions in different architectures and instruction sets. You are given 3 cases below. State the corresponding low level instructions using the *minimum* number of registers in each case. In all 3 cases, the "LOAD" and "STORE" instructions have the same semantics as shown in Table 1.

Table 1: "LOAD" and "STORE" instructions common to all 3 cases

INSTRUCTION		OPERANDS	SEMANTICS
LOAD	<OP1>,<OP2>	<OP1> can be only a register; <OP2> can be a register or the name of a variable in memory.	If <OP2> is a register, then its content is copied to the register of <OP1>; else the content of the named variable is copied from memory to the register of <OP1>.
STORE	<OP1>,<OP2>	<OP1> can be only a register; <OP2> is the name of a variable in memory.	The content of the register of <OP1> is copied to memory in the variable named in <OP2>.

Table 2: "MULT" and "ADD" instructions for the ISA of Case A

INSTRUCTION		OPERANDS	SEMANTICS
MULT	<OP1>,<OP2>	<OP1> can be only a register; <OP2> can be a register or the name of a variable in memory.	<OP1> = <OP1> x <OP2> where the multiplication uses either the contents of the two registers of <OP1> and <OP2>, or it uses the content of the register in <OP1> and the content of the named variable of <OP2>. In both cases the result is assigned to the register in <OP1>.
ADD	<OP1>,<OP2>	<OP1> can be only a register; <OP2> can be a register or the name of a variable in memory.	<OP1> = <OP1> + <OP2> where the addition uses either the contents of the two registers of <OP1> and <OP2>, or it uses the content of the register in <OP1> and the content of the named variable of <OP2>. In both cases the result is assigned to the register in <OP1>.

[2] **CASE A:** state the steps to execute "A = B + C x D;" using the ISA of Tables 1 and 2.

Table 3: "MULT" and "ADD" instructions for the ISA of Case B

INSTRUCTION		OPERANDS	SEMANTICS
MULT	<OP1>,<OP2>	Both <OP1> and <OP2> must be registers.	<OP1> = <OP1> x <OP2> where the multiplication uses the contents of the two registers of <OP1> and <OP2>. The result is assigned to the register in <OP1>.
ADD	<OP1>,<OP2>	<OP1> can be only a register; <OP2> can be a register or the name of a variable from memory.	<OP1> = <OP1> + <OP2> where the addition uses either the contents of the two registers of <OP1> and <OP2>, or it uses the content of the register in <OP1> and the content of the named variable of <OP2>. In both cases the result is assigned to the register in <OP1>.

[2]**CASE B:** state the steps to execute "A = B + C x D;" using the ISA of Tables 1 and 3.

Table 4: "MULT" and "ADD" instructions for the ISA of Case C

INSTRUCTION		OPERANDS	SEMANTICS
MULT	<OP1>,<OP2>	Both <OP1> and <OP2> must be registers.	<OP1> = <OP1> x <OP2> where the multiplication uses the contents of the two registers of <OP1> and <OP2>. The result is assigned to the register in <OP1>.
ADD	<OP1>,<OP2>	Both <OP1> and <OP2> must be registers.	<OP1> = <OP1> + <OP2> where the addition uses the contents of the two registers of <OP1> and <OP2>. The result is assigned to the register in <OP1>.

[2] **CASE C:** state the steps to execute "A = B + C x D;" using the ISA of Table 4.

Question 11. [8]

(a) [5] Number the following steps from 1 to 5 in the order they are performed in processing a general interrupt sequence using the interrupt jump table technique.

	Recognize the interrupt event and set the event flag
	Load the PC with the address from the interrupt vector table
	Execute the first instruction of the interrupt handling routine
	Push the processor registers onto the stack
	Determine the interrupt vector number

(b)1] Explain briefly the necessity for "levels" of interrupts.

(c)2] After an exception or interrupt has been processed, it may be the case that normal processing can resume within the application. State two examples in which this is not the case and what you think it actually happens.

Exmple

Question 12. [10] The C language provides two different syntactic forms for accessing the elements of an array. The alternatives are illustrated by the following two functions which identify the minimum value in a (non-empty) sequence of integers, where integers are 4 bytes in size.

```
int Min1(int* A, int N) {
   int *end, min = *A;
   for ( end = A+N; A < end; A++ ) {
    if ( *A < min ) min = *A;
   }
   return min;
}
```

```
int Min2(int* A, int N) {
  int i, min = A[0];
  for ( i = 0; i < N; i++ ) {
    if ( A[i] < min ) min = A[i];
  }
  return min;
}
```

(a) [1] What is the C datatype of A in `Min1`? Circle the most appropriate answer below:

[i] `int`	**[ii]** array of `int`	**[iii]** pointer to `int`	**[iv]** pointer to array of `int`

(b) [1] Is the datatype of A different in `Min2`? Explain:

(c) [3] Assuming that `A`, `N`, `end` and `min` are held in registers `R0` through `R3` respectively, give ARM assembly code which corresponds *exactly* to the statement: `if(*A < min) min = *A;` in `Min1`.

(d) [3] Assuming that `A`, `N`, `i` and `min` are held in registers `R0` through `R3` respectively, give ARM assembly code which corresponds *exactly* to the statement: `if(A[i] < min) min = A[i];` in `Min2`.

(e) [2] Which of the two functions, **`Min1`** or **`Min2`**, would you expect to execute faster for values of `N` > 1? Give at least one concrete reason why you think one is faster than the other, or why both should yield the same performance.

Question 13. [8] Assume you have a system which contains almost all the elements which have been discussed in the course, namely Virtual Memory, Cache, Pipelining, MMU (Memory Management Unit), TLB (Translation Lookaside Buffer), Page Table, DMA (Direct Memory Access). You need to explain how all these function together when the processor needs to read data and produces a virtual address which needs to be translated into a physical address. You are expected to write a cogent, brief, clear and precise explanation to following guiding questions: What is Virtual Memory? What are the steps which take place when there is a cache hit (best scenario)? What are the steps which take place when the needed data is only on disk (worst scenario)?

Question 14. [8] In an ARM program a 2-dimensional array has been allocated and initialized such that it contains the data shown in Figure 3, both in its 2-dimensions programmer's view and in its memory layout view. The notation *"e"* is placed in elements unused at this point. Register R1 contains the address of this data structure. The 2-dimensional array has maximum size of 3 rows and 6 columns, yet at this point only a 3x4 subset is occupied by integers, with the rightmost 2 columns remaining unitialized. Register R3=6 contains the total number of columns for this array, while register R4=4 denotes the number of columns currently used. A second 1-dimensional array also exists, also shown in Figure 3, with 6 elements, all unitialized,with its starting address in R5.

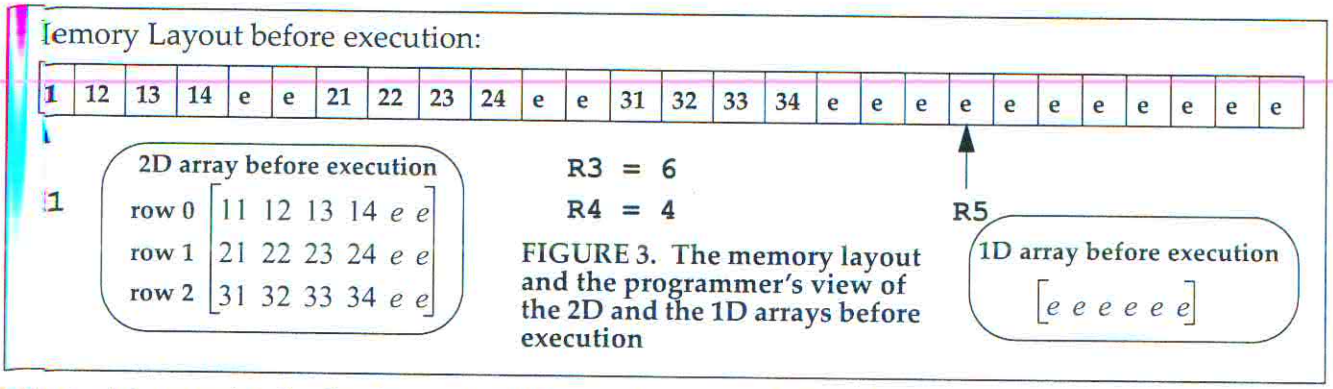

Iemory Layout before execution:
1 | 12 | 13 | 14 | e | e | 21 | 22 | 23 | 24 | e | e | 31 | 32 | 33 | 34 | e | e | e | e | e | e | e | e | e | e | e
1
2D array before execution
row 0 [11 12 13 14 e e]
row 1 [21 22 23 24 e e]
row 2 [31 32 33 34 e e]
R3 = 6
R4 = 4
R5
1D array before execution
[e e e e e e]

FIGURE 3. The memory layout and the programmer's view of the 2D and the 1D arrays before execution

Your task is to write the few lines of ARM code to copy *only the elements currently used in a row* of the 2-dimensional array to the 1-dimensional array. For this instantiation of the code, register R2=1 denotes the row number to be copied. The expected final output is shown in Figure 4.

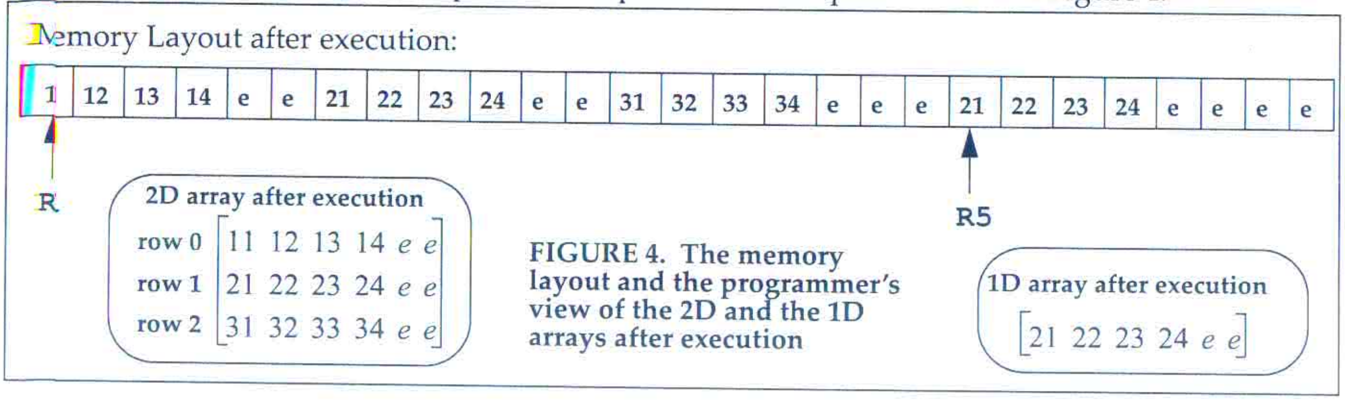

Memory Layout after execution:
1 | 12 | 13 | 14 | e | e | 21 | 22 | 23 | 24 | e | e | 31 | 32 | 33 | 34 | e | e | e | 21 | 22 | 23 | 24 | e | e | e | e
R
2D array after execution
row 0 [11 12 13 14 e e]
row 1 [21 22 23 24 e e]
row 2 [31 32 33 34 e e]
R5
1D array after execution
[21 22 23 24 e e]

FIGURE 4. The memory layout and the programmer's view of the 2D and the 1D arrays after execution

THE END