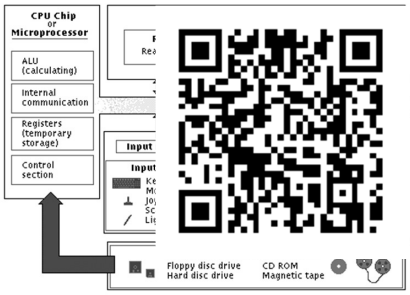
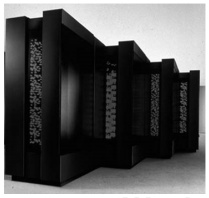


COMP25111
Operating Systems
Lectures 15:
Controlling Input and Output 2

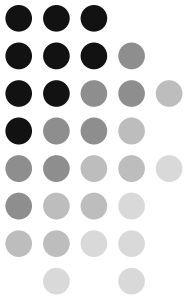
Dr Richard Neville
r.neville@manchester.ac.uk
Room: G12 Kilburn Building, Bottom floor



Week 9

NOTE: The **up-to-date** version of this lecture is kept on the associated web site – available [on-line] @ Blackboard select: COMP25111 Introduction to Computer Systems **www.manchester.ac.uk/portal**

http://www.cs.man.ac.uk/~neville/COMP25111/Lecture15/Lecture15.html



Where to find this Lecture 10 of the COMP25111 course?

First Go to Blackboard 9; then select: COMP25111 Operating Systems

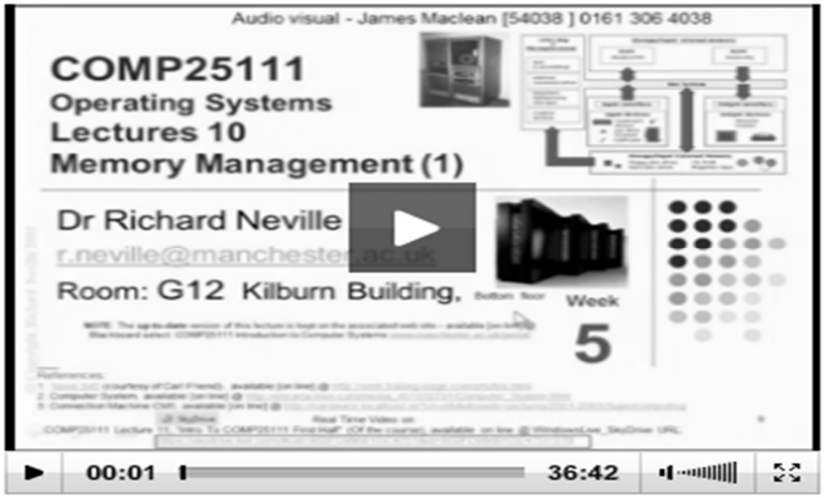
Then select: **Week 7**

This topic provides...
10: Memory management 1 (Introduction to basics) by RN;
11: Memory management 2 (Virtual Memory (1)) by RN;

Then select: **Lecture 10 Information**

Then select: **Real Time Video of Lecture 10**

Then select:



THIS week's Short Exam Questions Q1

1. Question

   List a number of devices that connect to the system bus, via I/O modules.

   ANSWER(S):

   - Answer(s):

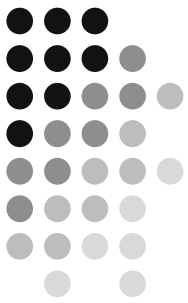
- NOTE: In the exam approximately 2 question are taken from the topics (and program examples) coved in each lecture

COMP25111

Systems Architecture 1

Lectures 15

Controlling Input and Output 2



Learning Outcomes

D-words

C-words

You should be able to:

1) Describe what a processor does in response to an **interrupt;**
2) Discuss what is meant by interrupt-driven I/O; &
3) Explain what is meant by direct memory access and how it is useful in I/O;
4) Exams and Revision; and
5) L3 Week 12 Hints.

FOOTNOTE2: D-words: direction words. C-Words: content words.
Ref.: Michael J. Wallace (1980, 2004)
Study Skills in English, ISBN 9780521537520.
D-Words also aligned to; Ref.: Taxonomy of Educational Objectives: The Classification of Educational Goals; pp. 201–207; B. S. Bloom (Ed.) Susan Fauer Company, Inc. 1956.

Footnote
1: Describe: aligned to paraphrase, discuss & give example [pictorial or diagrammatic].
2: Discuss: aligned to describe, paraphrase, discuss & give example..
3: Explain: aligned to describe; discuss; give examples.

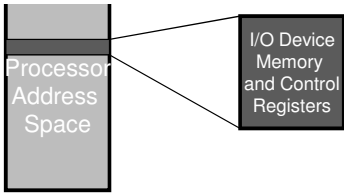
Reference: Bloom, B.S., *Taxonomy of Educational Objectives*. Handbook I: The Cognitive Domain. 1956: New York: David McKay Co Inc.

Interrupts

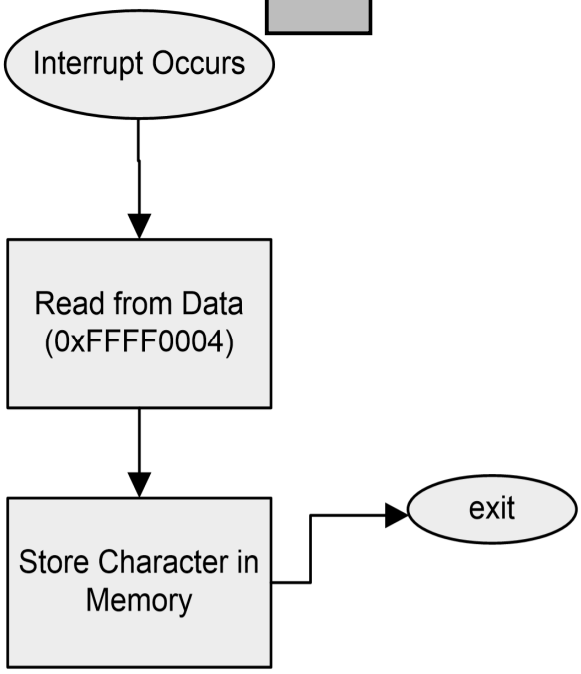
- Interrupts have a wide variety of uses.
- We will focus on a hardware interrupt generated by an external device.
- So, let's give our keyboard a connection to our processor's interrupt line.
- Our keyboard will send an interrupt (change a wire from '0' to '1') when it has a character ready to send.

Flow Chart with Interrupts

A few additions to notes are required.



- There is **no need** to read the **status register**, the interrupt indicates that a character has arrived, so we can simply read from the **data register**.
- Note that every time a character arrives the processor will be
- There is no time wasted checking the status register when no character has arrived.



References:

1: *Lecture* 9. Review Question. 2. How is *Memory mapped IO* used in ECE412. The PCMCIA attribute memory and common memory is mapped into the kernel space by the **...**, available [on line] @ *www.cs.rice.edu/~gw4314/**lecture**s/sram-dram.ppt*, (Last date accessed 28-11-2010).

What happens in an interrupt? (1)

Steps the processor takes:

1) The I/O device will signal that an interrupt has occurred by using the **interrupt line**;
2) The processor finishes executing its current instruction; and then
3) The processor then sends a special type of memory read (called an interrupt acknowledgement – **IACK**);

- The device that has interrupted the processor will respond with a value that indicates its identity.
  - This is required because several devices may need to interrupt the processor, the processor needs to know **which** device has sent the interrupt – the code that is run in response to an interrupt from the keyboard is different to that to handle an interrupt from a hard disk.

What happens in an interrupt? (2) Cont.

4) The processor saves the current value of the Program Counter and the contents of its registers into memory.

- The registers are memory locations inside the processor that it uses for [fast] calculations.
  - It [the registers] are a bit like the use of the output of a calculator but with several rather than just one.
- The value of the PC and registers are stored so that after the interrupt has been serviced, the program continue from the point at which it was interrupted.

What happens in an interrupt? (3) Cont.

5) The value that the processor received from the IACK cycle identifies what device interrupted the processor;
   - This is known as the interrupt vector.
6) The interrupt vector is used to access a table that holds the starting addresses of all programmes that handle interrupts;
   - The interrupt vector is used so that we get the starting address for the appropriate device.
   - This program is called an Interrupt Service Routine (ISR) and contains code that handles the type of interrupt.
7) This starting address of the ISR is loaded into the PC.
8) This code is now executed until a special instruction (return from interrupt) is executed;
   - This marks the end of the Interrupt Service Routine.

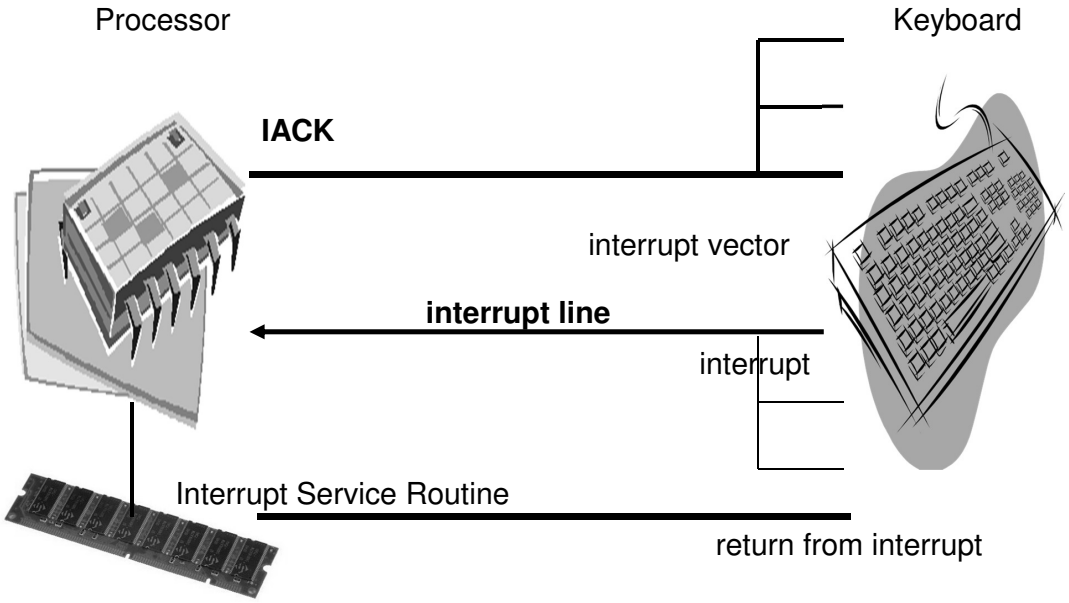
What happens in an interrupt? (4) Cont.

9) When the return from interrupt instruction is reached;
   - The PC and registers that were stored earlier are loaded back into the processor.

10) The program that was executing earlier continues as if nothing has happened.

- The character that has been placed into memory may be used later – or rather than place it in memory, the ISR might have processed it in some way.

Diagrammatic interrupt sequence:



Interrupt summary

interrupt **ack**nowledgement – IACK

This summary sequence may be worth remembering…

1) External line interrupts processor;
2) IACK cycle identifies the interrupting device;
3) Processor accepts interrupt after current instruction;
4) The processor stores the information necessary to restart the original program following the interrupt;
5) ISR is run for interrupting device until return from interrupt instruction is reached; then finally…
6) Stored information is reloaded into the processor, processor continues executing the original program as if nothing had happened.

Interrupt Service Routine (ISR)

Interrupts in I/O

- The interrupt is very useful in our keyboard example, rather than periodically checking to see whether a key has been typed by reading the status register, the processor only executes code to get characters when one has been typed;
  - This removes the need to periodically check the status register of the I/O device.
- Interrupts are even more useful in the case of mass storage devices.

Interrupts in I/O

- Suppose that a processor wants to write to a disk –this may take as much as 15ms before the sector is located;
  - This process can be speeded up utilising direct memory access (DMA).
- **The 4-steps DMA process is:**

1. Processor inform the disk DMA I/O - to write data
   - by writing to a **command register** in the DMA I/O device;
2. DMA controller starts write process;
3. Process gets on with another work [process] until;
4. DMA device is finished;
   - which is signalled by an **interrupt**.
     - [This explicitly is where the **interrupt I/O** comes into play…]

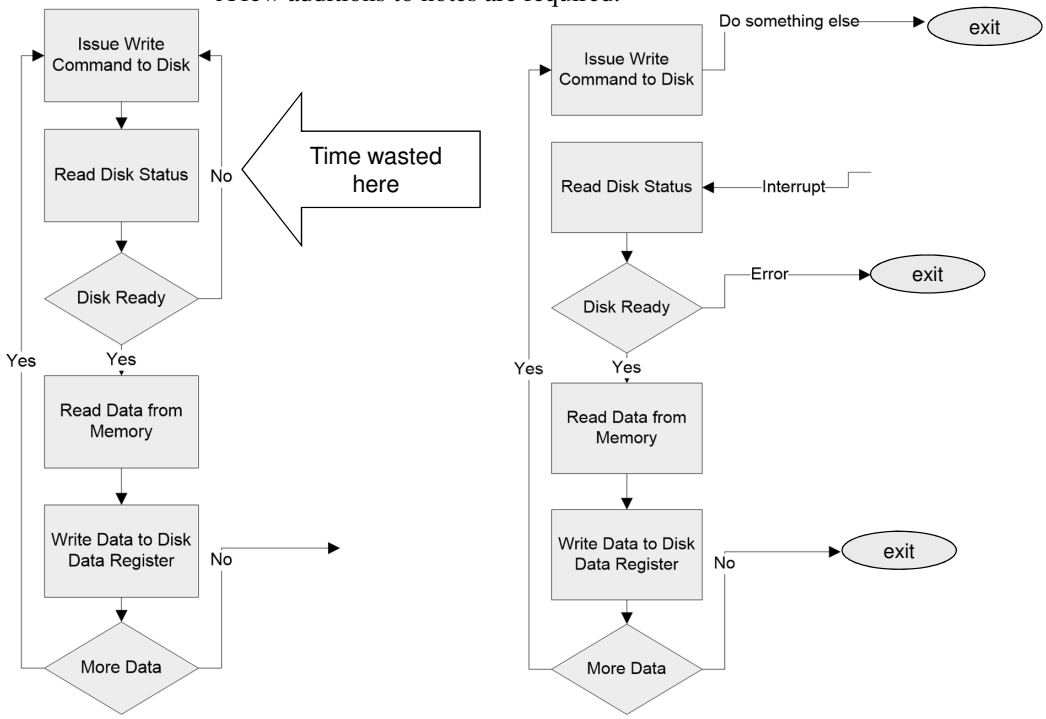
Flowcharts

- The next slide shows flowcharts for using programmed and interrupt-driven I/O for a disk where multiple writes are required;
  - i.e. we will be writing several multi-byte words, requiring several data transfers.
- We assume the disk I/O has a:

1) Command register where commands such as:
   - READ, WRITE, FORMAT can be written;
2) Status register that indicates whether the disk is ready for the data or not;
   - This is similar to the status register in the keyboard example.

Disk writing flowcharts

A few additions to notes are required.



Polling
Programmed I/O

Summary

- Interrupt-driven I/O is more efficient than programmed I/O, less time is wasted;
  - But both schemes rely on the processor to transfer the data; &
  - Data rate is limited by the processor's ability to read/write to/from the I/O device.
- For large volumes of data (e.g. disks) Direct Memory Access is performed.

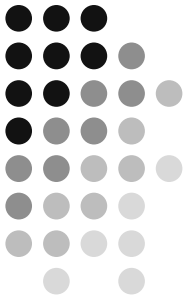
COMP25111

Systems Architecture 1

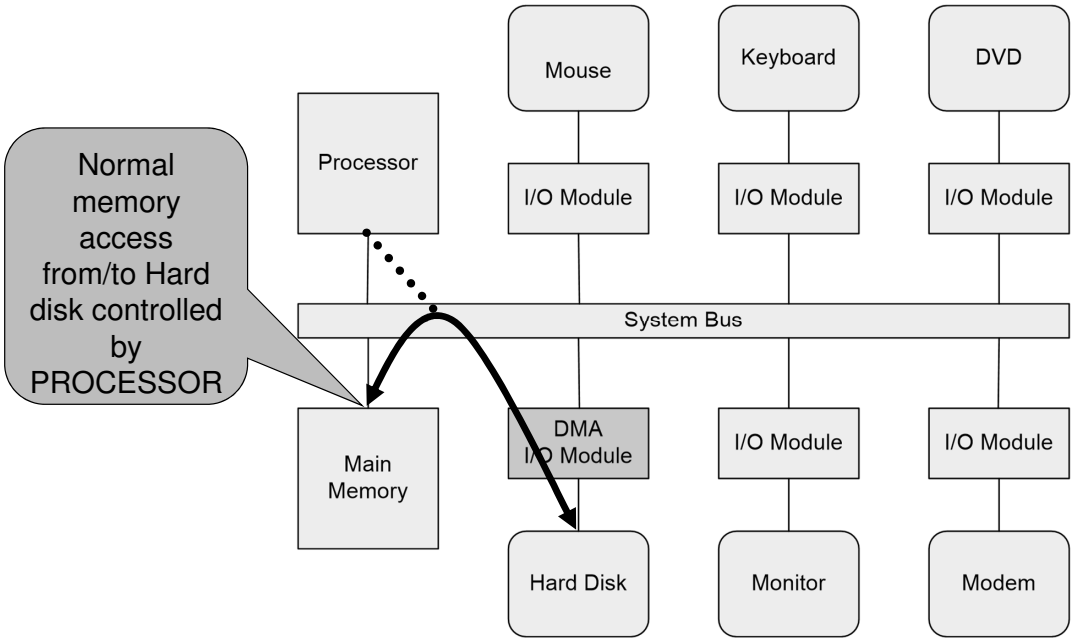
Lectures 15 Cont …

DMA

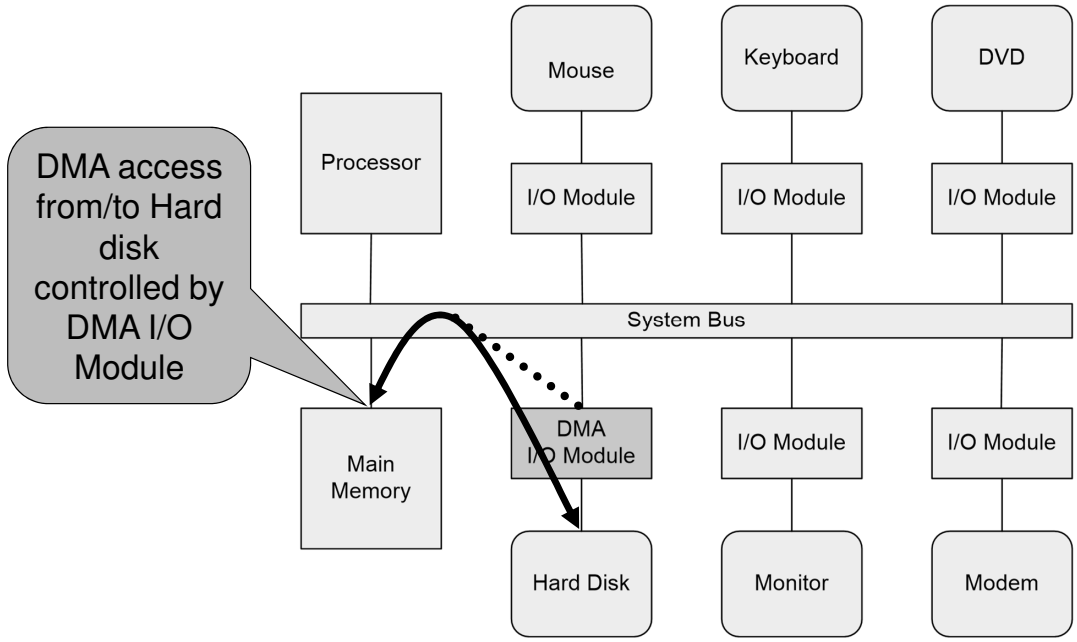
Direct Memory Access



Normal I/O



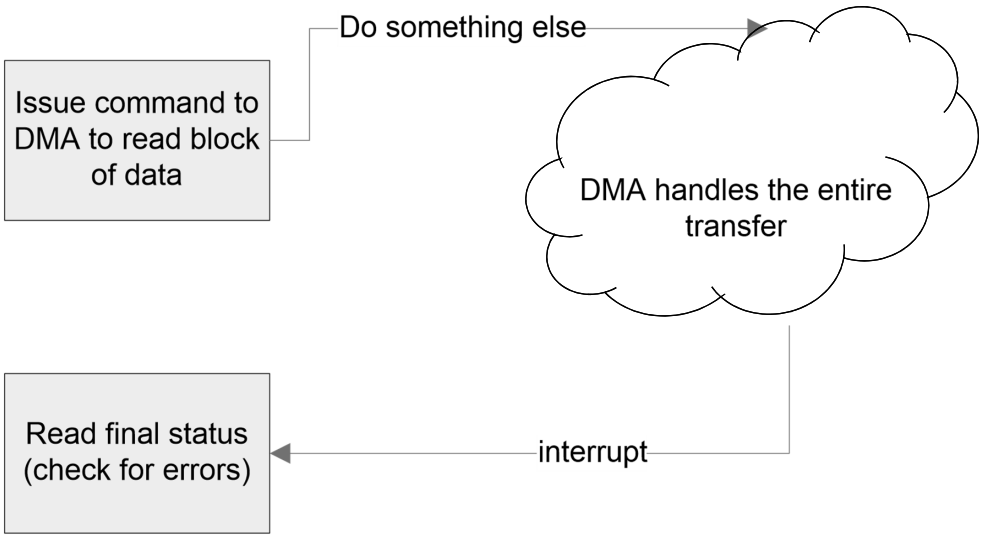
DMA I/O module

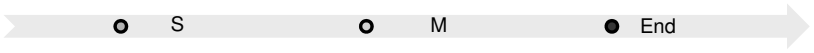


DMA

- A DMA device is capable of reading from or writing directly to memory in the same way as a processor does.
- It is optimised for transferring blocks of data into the memory system.
- In our example of reading from a disk, the DMA will handle the transfer of a whole block of data **without** processor intervention.

DMA Flowchart





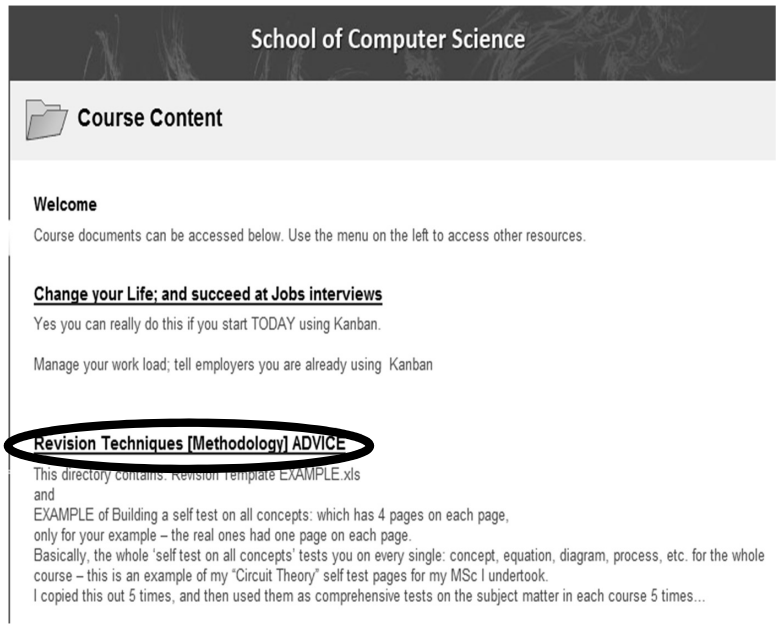
END

Revision Guidance

- See
- Blackboard 9

Why Didn't Someone Tell Me How To Revise

- If you do not have a comprehensive revision
- Methodology.
- Try the:
- Located on Blackboard 9.
- In the assessment directory.



Why Didn't Someone Tell Me How To Revise

- If you do not have a comprehensive revision
- Methodology.
- Try the:
- Located on Blackboard 9.
- In the assessment directory.

Revision Template EXAMPLE.xls

ExampleOfBuildaSelfTestOnAllConcepts.pdf

Exams and Revision

- Help a
  - Black
    In c

Revisior

Example .pdf

Why Didn't Someone Tell Me How To Revise

Specific revision guidance for COMP1702 Introduction to Computer Systems

0/ Re-listen to all lectures - boot up Audio recoding for lecture - then select lecture
- click through lecture while listening to Lecture audio...
1/ Re-read lecture notes THEN Undertake Q3's for each lecture
2/ Visit hardest concept(s) THEN Undertake all questions at the end of each lecture
3/ Compile (your own) summary sheets for each lecture
4/ Compile list of keywords for each lecture
5/ compile mind maps for each lecture
6/ Go through past exam - see module guide =- past exam in module guide [at end]
7/ Compile a possible future exam (virtual) - using pattern of last - sit un-timed
8/ Sit virtual exam timed
9/ Build self test sheets - these are a set of blank pages that you write QUESTION and ask yourself
- the concepts - the equations - the theorg - for all concepts and theorg of whole course
- copy these out 5 time to test yourself on ALL the theorg the course covers

A comprehensive revision schedule is:
1/ reread & comprehend lecture notes
2/ Visit hardest subject and reread notes
3/ Compile summary sheets
4/ Compile keywords
5/ Compile Mind maps (you can miss this if you dont find them useful)
6/ Go through past exam papers
7/ Compile virtual exam sit un-timed
7/ Compile virtual exam sit timed
8/ Build self test on all concepts

Note number 8/ is the hardest
- as you write out pages of info. that you should know - with blanks
for the answers - then copy the pages 5 times - make sure you know
all the concepts - using these tests - if you cannot answer one
- first: go to 4/ - see if these prompt you
- second: go to 3/ - see if these help
- thirdly: go to 1/
If you can go through one of the copies with no mistakes
- you have learnt all the concepts - but really these are just an aide-mémoire

6/ Should stop you building a false view of what you know
- as it should test you on all the concepts - this is a time consuming process
- but you then have build a self test that is a type of rote learning technique.

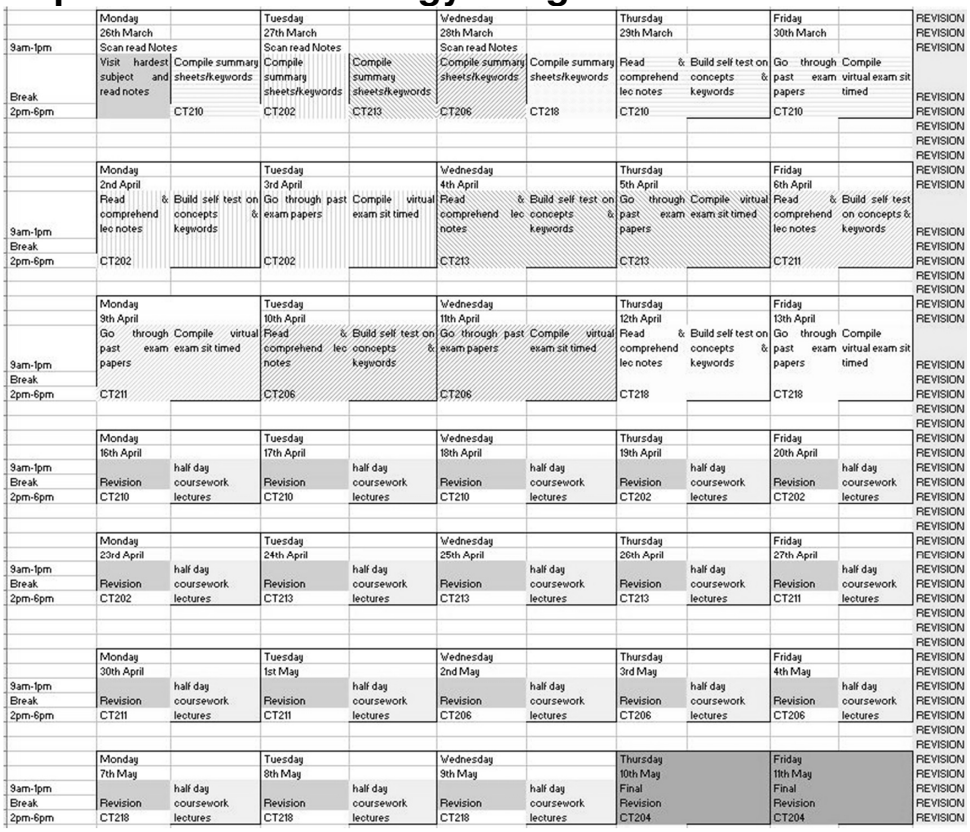
I hope the above helps
- I used these techniques when I undertook my MSc
- which had 8 subject per sem.
- all of which I had to comprehend fully.

THE exam time are PROVISIONAL

	Monday 26th March	Tuesday 27th March	Wednesday 28th March	Thursday 29th March	Friday 30th March	
9am-1pm	Scan read Natter	Scan read Natter	Scan read Natter			REVISION
	Visit hardest subject and reread notes	Compile summary sheets/keywords	Compile summary sheets/keywords	Read & comprehen dlecnotes	Build self test on concepts & keywords	Go through past exam papers	Compile virtual exam sit timed	REVISION
	Compile summary sheets/keywords	Compile summary sheets/keywords	Compile summary sheets/keywords					REVISION
Break								REVISION
2pm-6pm	CT210	CT202	CT213	CT206	CT210	CT210	CT210	REVISION
								REVISION
								REVISION

Learning; comprehension; & introspection

A possible methodology: Stage 1 - Revision schedule:



	Monday		Tuesday		Wednesday		Thursday		Friday		
	26th March		27th March		28th March		29th March		30th March		REVISION
9am-1pm	Scan read Notes		Scan read Notes		Scan read Notes						REVISION
	Visit hardest subject and read notes	Compile summary sheets/keywords	Compile summary sheets/keywords	Compile summary sheets/keywords	Compile summary sheets/keywords	Compile summary sheets/keywords	Read & comprehend lec notes	Build self test on concepts & keywords	Go through past exam papers	Compile virtual exam sit timed	
Break											REVISION
2pm-6pm		CT210	CT202	CT213	CT206	CT218	CT210		CT210		REVISION
											REVISION
											REVISION
											REVISION
	Monday		Tuesday		Wednesday		Thursday		Friday		REVISION
	2nd April		3rd April		4th April		5th April		6th April		REVISION
	Read & comprehend lec notes	Build self test on concepts & keywords	Go through past exam papers	Compile virtual exam sit timed	Read & comprehend lec notes	Build self test on concepts & keywords	Go through past exam papers	Compile virtual exam sit timed	Read & comprehend lec notes	Build self test on concepts & keywords	
9am-1pm											REVISION
Break											REVISION
2pm-6pm	CT202		CT202		CT213		CT213		CT211		REVISION
											REVISION
											REVISION
	Monday		Tuesday		Wednesday		Thursday		Friday		REVISION
	9th April		10th April		11th April		12th April		13th April		REVISION
	Go through past exam papers	Compile virtual exam sit timed	Read & comprehend lec notes	Build self test on concepts & keywords	Go through past exam papers	Compile virtual exam sit timed	Read & comprehend lec notes	Build self test on concepts & keywords	Go through past exam papers	Compile virtual exam sit timed	
9am-1pm											REVISION
Break											REVISION
2pm-6pm	CT211		CT206		CT206		CT218		CT218		REVISION
											REVISION
											REVISION
	Monday		Tuesday		Wednesday		Thursday		Friday		REVISION
	16th April		17th April		18th April		19th April		20th April		REVISION
9am-1pm		half day coursework lectures		half day coursework lectures		half day coursework lectures		half day coursework lectures		half day coursework lectures	REVISION
Break	Revision		Revision		Revision		Revision		Revision		REVISION
2pm-6pm	CT210		CT210		CT210		CT202		CT202		REVISION
											REVISION
											REVISION
	Monday		Tuesday		Wednesday		Thursday		Friday		REVISION
	23rd April		24th April		25th April		26th April		27th April		REVISION
9am-1pm		half day coursework lectures		half day coursework lectures		half day coursework lectures		half day coursework lectures		half day coursework lectures	REVISION
Break	Revision		Revision		Revision		Revision		Revision		REVISION
2pm-6pm	CT202		CT213		CT213		CT213		CT211		REVISION
											REVISION
											REVISION
											REVISION
	Monday		Tuesday		Wednesday		Thursday		Friday		REVISION
	30th April		1st May		2nd May		3rd May		4th May		REVISION
9am-1pm		half day coursework lectures		half day coursework lectures		half day coursework lectures		half day coursework lectures		half day coursework lectures	REVISION
Break	Revision		Revision		Revision		Revision		Revision		REVISION
2pm-6pm	CT211		CT211		CT206		CT206		CT206		REVISION
											REVISION
											REVISION
	Monday		Tuesday		Wednesday		Thursday		Friday		REVISION
	7th May		8th May		9th May		10th May		11th May		REVISION
9am-1pm		half day coursework lectures		half day coursework lectures		half day coursework lectures	Final		Final		REVISION
Break	Revision		Revision		Revision		Revision		Revision		REVISION
2pm-6pm	CT218		CT218		CT218		CT204		CT204		REVISION

Learning; comprehension; & introspection

A possible methodology: Stage 1 - Revision schedule: In detail:

Monday		Tuesday		Wednesday		Thursday		Friday	
26th March		27th March		28th March		29th March		30th March	
Scan read Notes		Scan read Notes		Scan read Notes					
Visit hardest subject and read notes	Compile summary sheets/keywords	Compile summary sheets/keywords	Compile summary sheets/keywords	Compile summary sheets/keywords	Compile summary sheets/keywords	Read & comprehend lec notes	Build self test on concepts & keywords	Go through past exam papers	Compile virtual exam sit timed
	CT210	CT202	CT213	CT206	CT218	CT210		CT210	

Monday		Tuesday		Wednesday		Thursday		Friday	
2nd April		3rd April		4th April		5th April		6th April	
Read & comprehend lec notes	Build self test on concepts & keywords	Go through past exam papers	Compile virtual exam sit timed	Read & comprehend lec notes	Build self test on concepts & keywords	Go through past exam papers	Compile virtual exam sit timed	Read & comprehend lec notes	Build self test on concepts & keywords
CT202		CT202		CT213		CT213		CT211	

Monday		Tuesday		Wednesday		Thursday		Friday	
9th April		10th April		11th April		12th April		13th April	
Go through past exam papers	Compile virtual exam sit timed	Read & comprehend lec notes	Build self test on concepts & keywords	Go through past exam papers	Compile virtual exam sit timed	Read & comprehend lec notes	Build self test on concepts & keywords	Go through past exam papers	Compile virtual exam sit timed
CT211		CT206		CT206		CT218		CT218	

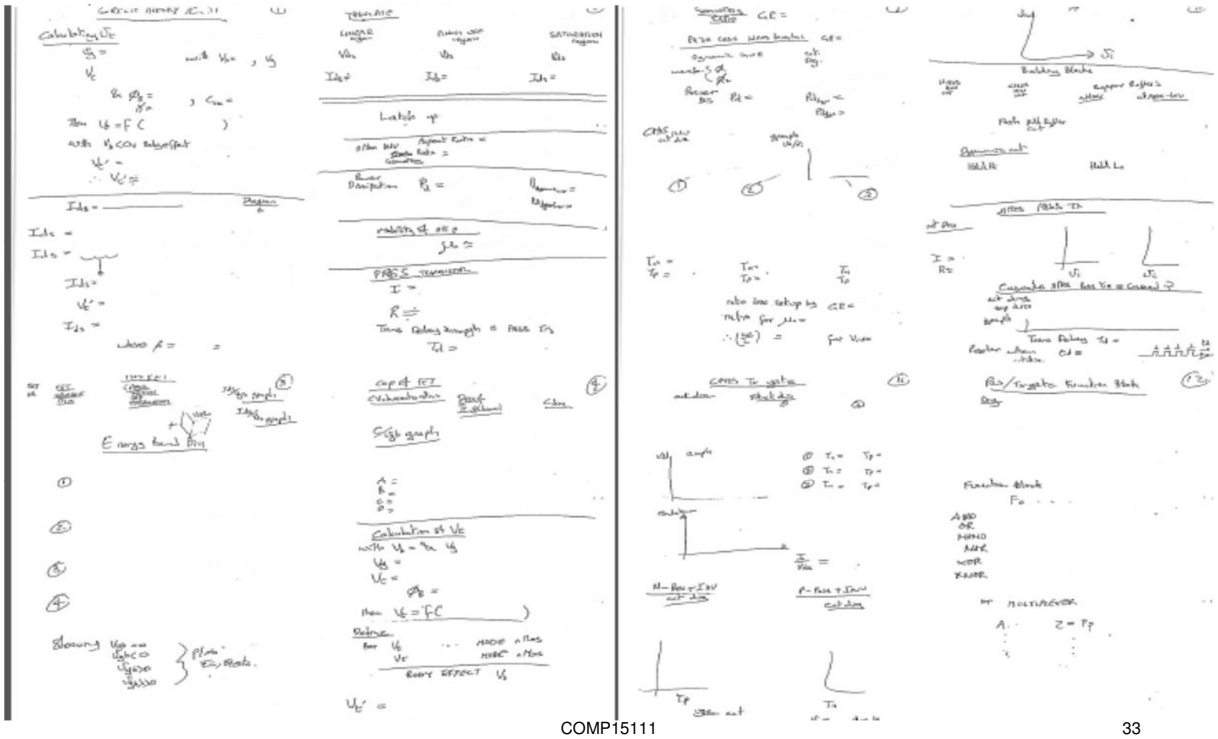
Monday		Tuesday		Wednesday		Thursday		Friday	
16th April		17th April		18th April		19th April		20th April	
Revision	half day coursework lectures	Revision	half day coursework lectures	Revision	half day coursework lectures	Revision	half day coursework lectures	Revision	half day coursework lectures
CT210		CT210		CT210		CT202		CT202	

A possible methodology: Stage 2 - Exam schedule:

	Monday		Tuesday		Wednesday		Thursday	Friday	REVISION
	7th May		8th May		9th May		10th May	11th May	REVISION
9am-1pm		half day		half day		half day	Final	Final	REVISION
Break	Revision	coursework	Revision	coursework	Revision	coursework	Revision	Revision	REVISION
2pm-6pm	CT218	lectures	CT218	lectures	CT218	lectures	CT204	CT204	REVISION
	Monday		Tuesday		Wednesday		Thursday	Friday	EXAMS
	14th May		15th May		16th May		17th May	18th May	EXAMS
9.30am-11.30am			Final		CT218	EXAM	Final	Final	EXAMS
Break			Revision				Revision	Revision	EXAMS
2pm-4pm	CT204	EXAM	CT218				CT206	CT206	EXAMS
									EXAMS
									EXAMS
	Monday		Tuesday		Wednesday		Thursday	Friday	EXAMS
	21th May		22nd May		23rd may		24th May	25th May	EXAMS
9.30am-11.30pm	CT206	EXAM	Final				Final		EXAMS
Break			Revision				Revision		EXAMS
2pm-4pm			CT211		CT211	EXAM	CT213	CT213 EXAM	EXAMS
									EXAMS
									EXAMS
	Monday		Tuesday		Wednesday		Thursday	Friday	EXAMS
	28th May		29th May		30th May				EXAMS
9.30am-11.30am	Final		CT202	EXAM	CT210	EXAM			EXAMS
Break	Revision								EXAMS
2pm-6pm	CT202 /	CT210							EXAMS

A possible methodology: Stage 3

Specific revision guidance for COMP15111 Introduction to Computer Systems
0/ Re-listen to all lectures - boot up Audio recoding for lecture - then select lecture
- click through lecture while listening to Lecture audio...
1/ Re-read lecture notes THEN Undertake Q3's for each lecture
2/ Visit hardest concept(s) THEN Undertake all questions at the end of each lecture
3/ Compile (your own) summary sheets for each lecture
4/ Compile list of keywords for each lecture
5/ compile mind maps for each lecture
6/ Go through past exam - see module guide =- past exam in module guide [at end]
7/ Compile a possible future exam (virtual) - using pattern of last - sit un-timed
8/ Sit virtual exam timed
9/ Build self test sheets - these are a set of blank pages that you write QUESTION and ask yourself
- the concepts - the equations - the theory - for all concepts and theory of whole course
- copy these out 5 time to test yourself on ALL the theory the course covers



Feedback From student whom used the method:

- Hello Richard,
-
- I hope you had great Christmas/New Year.
-
- XXXX
-
- *On a side note,* I'm using the 'self-assessment' technique you've recommended (for revision) and have written myself a 30 question
- *test for one of my modules which I do every other day. It's proving very useful!*
-
- Thanks,
- Xxxxxx
-
- The point here is:
- Is that a student has used the 'self-assessment' technique; and said:
- “It's proving very useful!”

More feedback

- Another - **Note great feedback from student, before Exam:**
- From the above student…
- With regards to #RN3, the self-assessment questions definitely allowed me to retain and recall large amounts of domain knowledge.
- It was especially useful in shorter questions and proved more beneficial than simply re-reading notes in a repetitive manner.
- To build/reinforce this knowledge I found it useful to answer questions/teach others, finding that if I could not sufficiently explain to a **peer** the concept,
- Then I did not truly understand it myself. I'm not a massive fan of group revision, but we organised several Q&A sessions in small groups which worked well.
- I think the final piece of the puzzle and something which is hard to 'revise', is *knowledge application*. For one of my modules I was extremely well prepared with my own self-assessment, past-papers and mock-questions; I therefore had a substantial amount of experience in applying my knowledge come the exam.
- However for another module there was a lack of available material with which to apply my knowledge, due to changes in the syllabus there was only one relevant past paper and no mock questions were provided by the lecturer (they chose not to, for fear of similarity to the actual exam!).
- Ultimately this meant repetition of notes and group sessions were the only real ways to prepare. Which meant I definitely wasn't as confident walking into this exam.
- If you have any suggestions for different techniques for revision given this kind of situation, that would be appreciated.

Important Exam advice, cont…

1. Write your answers on the exam paper in a big font. In previous exams, I have tried to decipher writing that was equivalent to 5 or 6 font – far too small. If you have trouble reading it, so will the examiner.
2. Take care writing in the answer book, so [at least] the answers and diagrams and mathematical working-out is readable, tidy, clear and written in good Plain English.
3. Students need to layout their answers, in their answer book, so each different answer is CLEARLY differentiated from the next. Leave a clear line between each answer.
4. If number conventions such as 2., then 2.a, then 2.a.ii PLEASE use them.
5. Please try to answer your questions in order.
6. Please explain any diagrams [or tables] you draw [up] in full. Full marks cannot be awarded unless full explanations are given to answer each question full and comprehensively.

BIG HINT – diagram really do help:

Remember, good – honours grade answer – in the exam – for a question – to maximise marks – should – or you should think of adding A DIAGRAM;

basic layout:

1) Textual answer;
2) Diagram supporting answer; &
3) Full explanation of diagram…

This sort of answer will maximise your marks…

…practice doing this for the majority of your answer – if you can align a diagram that is…

Where to find Lab. L3 Week 12 Hints – Final Lab (Page table)?

First Go to Blackboard 9; then select: COMP25111 Operating Systems

Then select:

Lab3_RN

Move to this folder, to download ex3:
Click "Lab3.pdf" to obtain (read) Lab3 (ex3) full specifications.
Plus Lab 3 Hint 1: [#H1]; Hint 2: [#H2] etc.
Note read the "Lab Rules" as you must use SUBMIT for Lab 1, 2 & 3
If you use Netbeans it is on all Lab PC :- Where to Find NetBeans on the lab machines:
Boot into Windows 7; 1) Click "Start;" 2) Click "All programs;" 3) Click the "NetBeans" folder [icon]; 4) Click the "NetBeans IDE 7.0.1" icon.

Then select:

Lab3 Hints

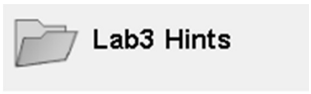
Hint 1: [#H1]
Hint 2: [#H2] etc.
If you use Netbeans it is on all Lab PC :- Where to Find NetBeans on the lab machines:
Boot into Windows 7; 1) Click "Start;" 2) Click "All programs;" 3) Click the "NetBeans" folder [icon]; 4) Click the "NetBeans IDE 7.0.1" icon.

Then select: see next page…

Learning; comprehension; & introspection

L3 Week 12 Hints

Sub directory:



H1.pdf

H2.pdf

H3.pdf

H3_2.pdf

H4.pdf

traceSingleFetch

traceSingleRead

traceSingleWrite

FIFOvisualExample.pptx

Learning; comprehension; & introspection

L3 Week 12 Hints

Sub directory:

#H1

'Hint No 1' hints at the requirements you should have data mined from reread section 5.

Stage 1: extracting the requirements of the laboratory from COMP20051 Exercise 3: Simulation of Paging Behaviour laboratory description document.

[#R1] Write a java program to implement a simple MMU.

Extracted from:

… Purpose of the exercise is to write Java code to implement a simple MMU, which contains a page table and code to perform the necessary actions when read or write accesses occur to a virtual memory address. …

[#R2] Implement a LRU algorithm.
[#R3] [Implement a] 'timestamp' in each page table entry.

Extracted from:

… You should implement a LRU algorithm to determine page rejection. To make this simple, you should have a 'timestamp' in each page table entry, which is updated appropriately when a page is accessed. …

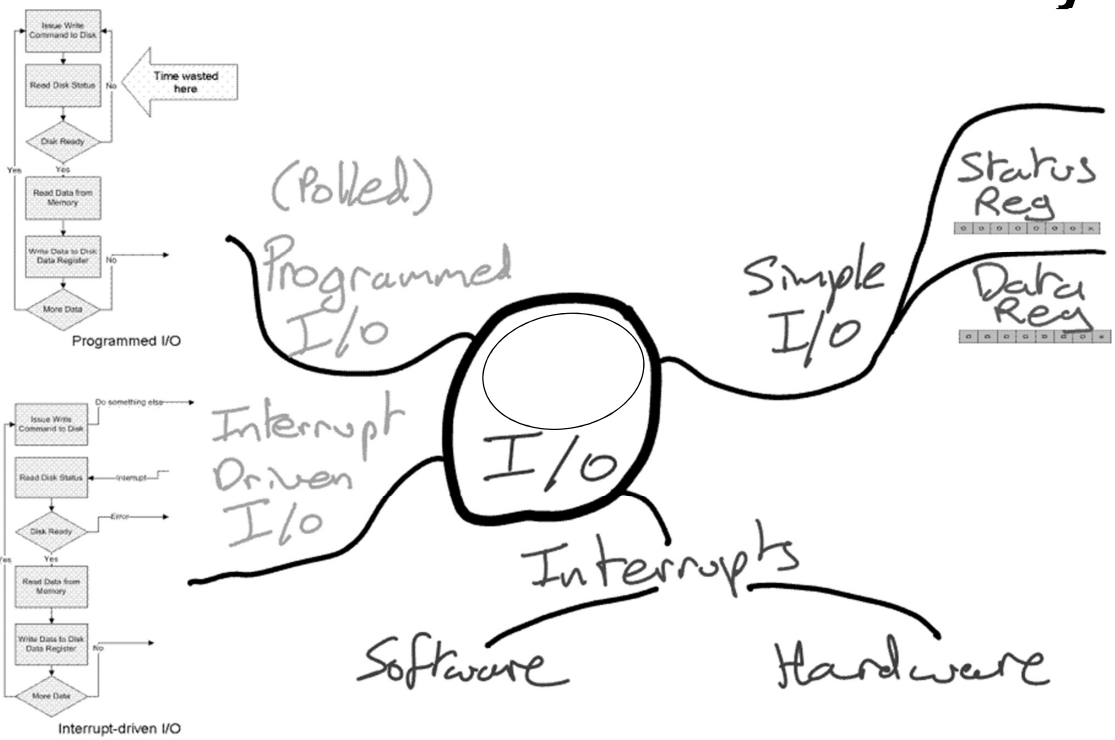
[#R4] [Implement] a 'dirty' indication in the page table.

Extracted from:

… You should also include a 'dirty' indication in the page table implementation, which is set on a write. This indicates if the page needs to be written back to disk if it is rejected. …

© Copyright Richard Neville 2009\10

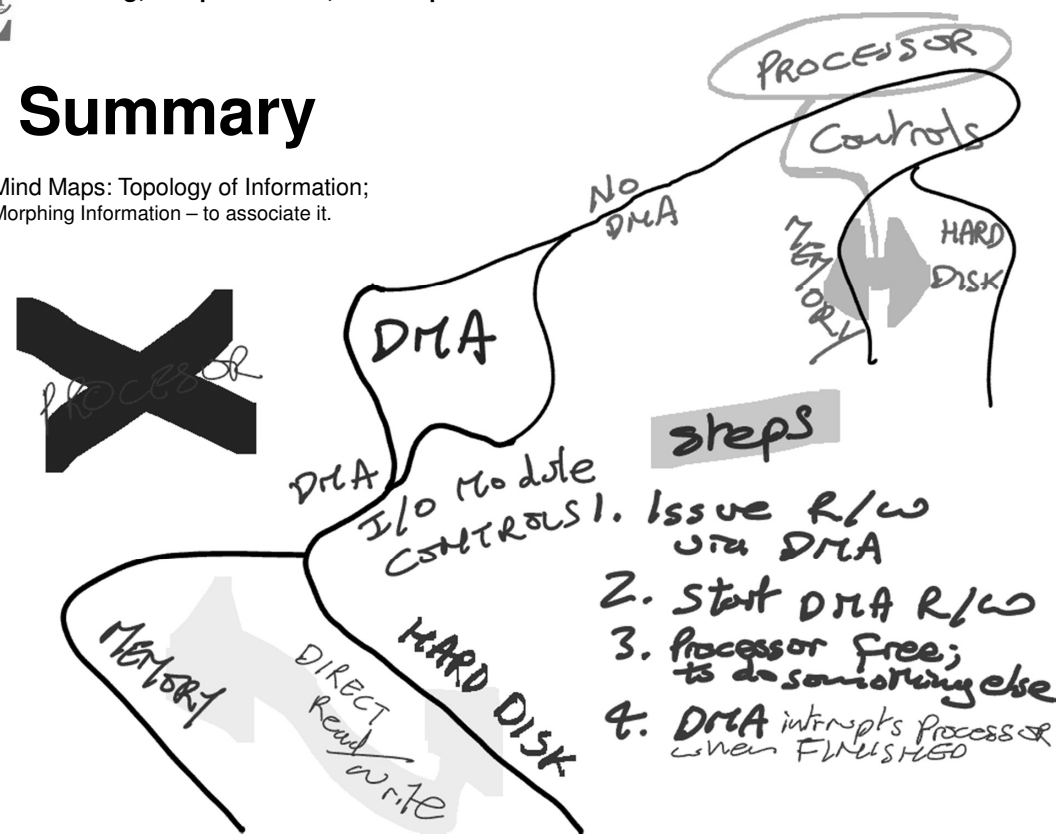
Summary



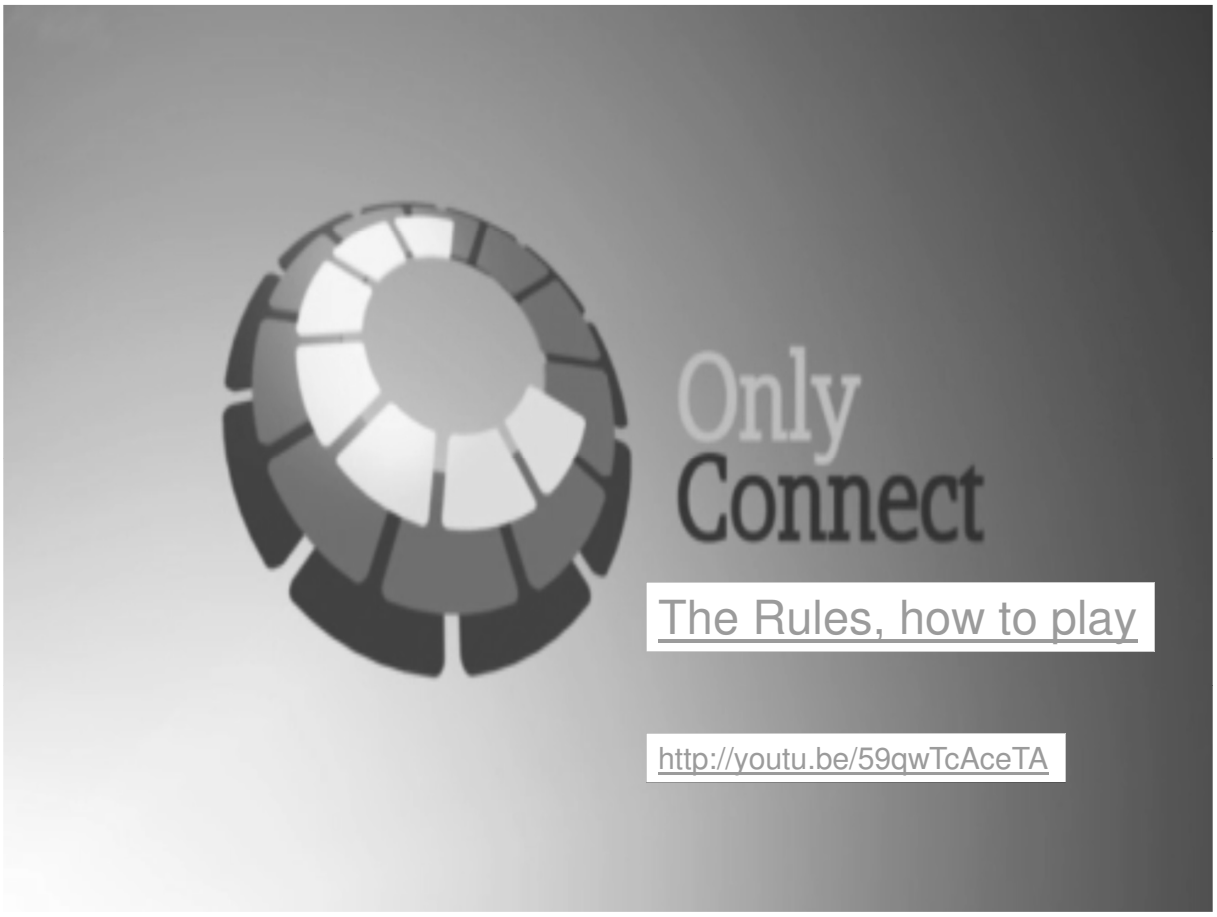
Mind Maps: Topology of Information;
Morphing Information – to associate it.

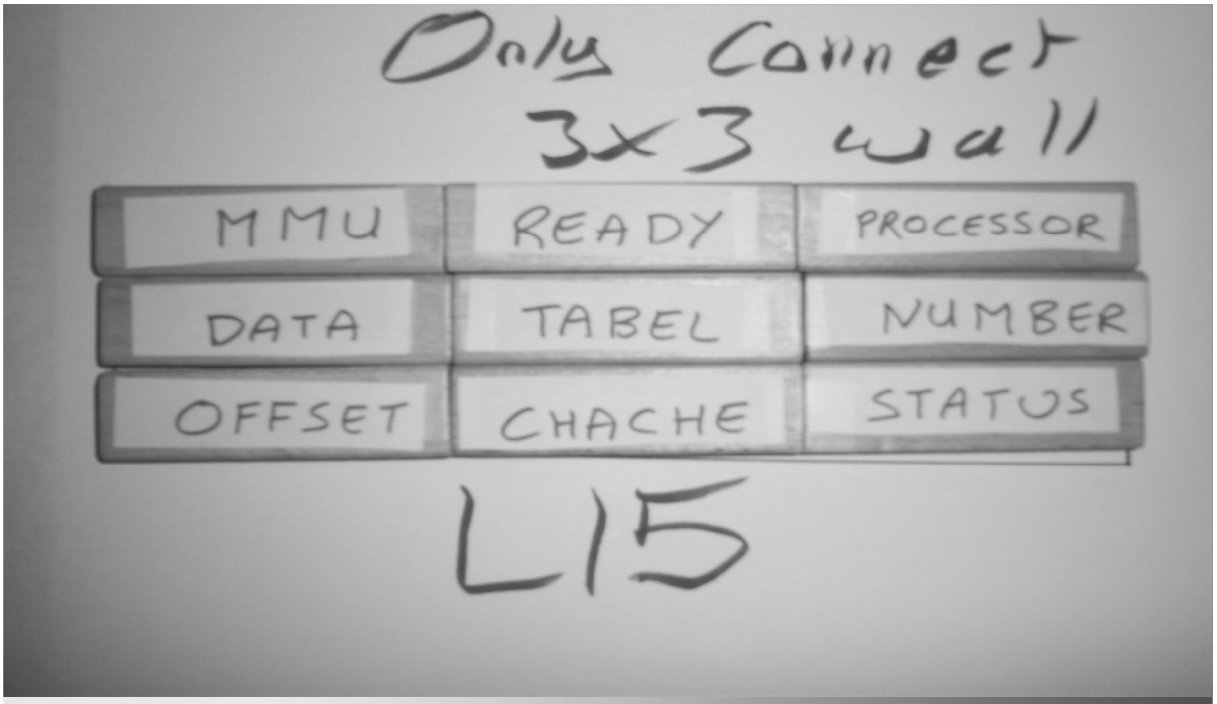
Summary

Mind Maps: Topology of Information;
Morphing Information – to associate it.




Only
Connect
Quiz


Only
Connect
The Rules, how to play
http://youtu.be/59qwTcAceTA



Quiz

List of Questions to ask lecturer

- Before the 9a.m. start lecture the lecturer will be half an hour early and you can ask [any and all] questions in that half hour; before the lecture:

1.

2.

3.

4.

5.

Learning; comprehension; & introspection

Getting ready for next week
Do next week's Q3's NOW

- Once you have re-read the lecture notes; and listened to the audio recording [while stepping through the PPT] of the lecture again:
- Please have a think about next week's Q3's
  - on the next page
- If you try to answer the Q3's now you will be in a much better position to recall the information.
- Once you have done this, transfer your answers to next weeks "Student [OWN answers] version" at the start of next weeks lecture.
  - YES this implies bringing the last weeks lecture notes to the next lecture …

Learning; comprehension; & introspection

THIS week's Short Exam Questions Q3

1. Question
   List a number of devices that connect to the system bus, via I/O modules.

Answer(s):

2. Question
   Name the two different types of information I/O registers store?

Answer(s):

3. Question
   Name the two different types of interrupts.

Answer(s):

- NOTE: In the exam approximately 2 question are taken from the topics (and program examples) coved in each lecture

NEXT Lecturers

Short Exam Questions **Q3**

1. Question
   List a number of devices that connect to the system bus, via I/O modules.

Answer(s):

2. Question
   Name the two different types of information I/O registers store?

Answer(s):

3. Question
   Name the two different types of interrupts.

Answer(s):

- NOTE: In the exam approximately 2 question are taken from the topics (and program examples) coved in each lecture

Glossary

• Why build a Glossary for each course unit you undertake?

- It is imperative that the correct terminology [keywords] are utilised in context in your exam answers; this is so important that the lecturer has added glossaries to each paper copy of your lectures. It is of such importance that in your notes [prior to the start of the glossary] the following advice is given:
  - Each module you undertake uses its own jargon.
  - This can be a problem for new students, whom are trying to comprehend the new domain knowledge attached to a particular new module.
  - One way to get to know the new jargon is to build your own GLOSSARIES for each course module.
  - The glossary on the next few pages is a starting point for this module [unit].
  - Please feel free to add to the glossaries throughout the unit…
  - The glossary is full of potential exam questions of the form "define X" or "briefly explain X."
- Please heed the advice in the future; even if your lecturers do not supply a glossary build your own as without knowledge of the appropriate terminology [keywords] when expounding your knowledge you will not be viewed as comprehending the details of any theory.

GLOSSARY

Using the on-line resources and any other resources compile a glossary of the terms below [PIPLINES: memory management]:

- Interrupt →
- Interrupt-driven →
- Acknowledgement →
- Vector →
- Interrupt vector →
- Command register →
- Status register →
- Polling →

Learning Resources 1

- **Descriptions [Theory] (in text books)**
- Remember the key issues, highlighted in GREEN, are the concepts to look for in any book:
  - Section on Interrupt, Vectored interrupt handling, Command register, Status register, Polling, in chapter 8 the Input-Output in: Chalk BS, Carter AT, Hind RW (2004) Computer Organisation and Architecture: An introduction 2nd Edition, Palgrave, ISBN 1-4039-0164-3 .
  - Section on Interrupt, Interrupt-driven, Acknowledgement, and Vectored Interrupt – in chapter 6 the Input/Output in: Computer Organization and Architecture, Fifth Edition by William Stallings.
  - Section on Interrupt – in a few chapters in: Structured Computer Organization, 5/E, Andrew S. Tanenbaum, Vrije University, Amsterdam, The Netherlands, ISBN-10: 0131485210, ISBN-13: 9780131485211, Publisher: Prentice.
  - Section on I/O hardware, Interrupt, Interrupt-handler, DMA, Software Interrupt vector in chapter 13 I/O systems in Operating System Concepts, A. Silberschatz et al, Wiley.
  - Section on Interrupt, Interrupt-driven, Acknowledgement, Interrupt vector, Polling in chapter 5 Input/Output in Modern Operating Systems, A. Tanenbaum, Prentice Hall.
- **Web resources:**
  - interrupt; available [on-line] @ http://www.webopedia.com/TERM/I/interrupt.html

Questions

Introduction to Questions:

The set of questions are based on lecture 15.

Answer Sheet will be given later in year and will contain the answers to these questions.

- Remember to find detailed and comprehensive answer you should [also] reference associated text books in the library.
- A reasonable starting place for associated book titles are:

1) This units 'module guide'; given to you in RN's first lecture – or on the web [Blackboard];
2) Those books mentioned in 'Background Reading;'
3) Those books [and web resources] mentioned in Learning Resources.

Questions

1. Question

- In the event an interrupt occurring; one process has to hand over to another. State the basic steps that have to happen before the second process starts.

- **Answer**

Questions

1. Answer

Answer(s):

Questions

2. Question

- Given the overall steps the processor takes to handle an interrupt can be summarised in six basic steps. List the six basic steps (in a interrupt summary).

- **Answer**

Questions

2. Answer

1)
2)
3)
4)
5)
6)

Answer(s):

Questions

3. Question

- Explain how a peripheral communicates with the CPU using interrupts.

- **Answer**

3. Answer

- The following points should be covered to some degree in the answer:

Answer(s):

Revision Exercises

- Scan read Lecture 15's Questions.
  - Answer Lecture 15's Questions
    - Particularly those questions you had difficulties with when you first tried them.

Background Reading

- [1] Computer Organisation and Architecture: An introduction 2nd Edition, Palgrave, ISBN 1-4039-0164-3, Chalk BS, Carter AT, Hind RW (2004).
  - Sections in chapter 8 the Input-Output in: Section on Interrupt, Vectored interrupt handling, Command register, Status register, Polling.
- [2] Computer Organization and Architecture, Fifth Edition by William Stallings.
  - Sections in chapter 6 the Input/Output in: Section on Interrupt, Interrupt-driven, Acknowledgement, and Vectored Interrupt.
- [3] Structured Computer Organization, 5/E, Andrew S. Tanenbaum, Vrije University, Amsterdam, The Netherlands, ISBN-10: 0131485210, ISBN-13: 9780131485211, Publisher: Prentice.
  - Section on Interrupt – in a few chapters.
- [4] Operating System Concepts, A. Silberschatz et al, Wiley.
  - Section on I/O hardware, Interrupt, Interrupt-handler, DMA, Software Interrupt vector in chapter 13 I/O systems.
- [5] Modern Operating Systems, A. Tanenbaum, Prentice Hall.
  - Section on Interrupt, Interrupt-driven, Acknowledgement, Interrupt vector, Polling in chapter 5 Input/Output.

COMP25111 Exercise 3:

Simulation of Paging Behaviour

Contains a copy of the "traceA" file.

Before you start this PLEASE read the multimedia PPTs and audios on the Blackboard 9 portal- on the Blackboard 9 Web Site; select: COMP25111 Operating Systems
https://www.portal.manchester.ac.uk/; then
1/ Go to COMP25111 Operating Systems 2011-12 1st Semester;
2/ Then Lab3_RN
3/ Then Lab3 Hints
4/ Then read through as many hints as you need; remember the methods, processes, naming conventions, and implicit theory outlined in the hints will be useful to you in your final year projects where you should think about using these techniques…

Duration: 1 Session

1. Learning Outcomes

On completion of this exercise, a student will:

- Have implemented the functionality of a simple Memory Management Unit (MMU) for a paged virtual memory system in Java.
- Have exercised the MMU using a trace file of memory accesses using a Java program which initialises the MMU and calls methods within it to perform read and write actions.
- Have produced statistics, which show the behaviour of the MMU with particular reference to page faults and page rejections.

2. Introduction

Virtual memory is a technique for providing a process with apparent access to the whole addressable memory space of a processor in circumstances where the real memory available may be considerably smaller. One advantages of the scheme is the process can use larger code and data sizes than would otherwise be possible. Another is the run-time layout of code and data can be considerably simplified if it does not have to fit into a confined space. This is particularly true for dynamic data whose size is unknown until run-time.

Virtual memory relies on the fact that, in the majority of programs, the use of both code and data exhibits both *spatial locality* and *temporal locality*. That is that at any point in time, the program is usually only using a small fraction of its code and data and that, over a short time period, the usage will not change significantly.

A virtual memory system therefore tries to keep currently used code and data in the (limited size) real memory of the system while the rest (if it exists) is kept on background storage such as magnetic disk. In order to relieve the programmer from the complex task of working out which data should be where at any point in time, the system arranges to move the data between real memory and disk automatically as required.

A piece of hardware know as a Memory Management Unit (MMU), together with system software within the Operating System provide all the functionality needed to

do this. The purpose of this exercise is to implement the functionality of a MMU in order to gain an understanding of the principles.

3. Paged Virtual Memory

Devices such as disks usually store and retrieve data in blocks of hundreds or thousands of bytes. Accessing a block is often slow compared to CPU memory speeds but once accessed; the contents of that block can be read or written very rapidly. In these circumstances, it makes sense to move data in units which are larger than bytes or CPU words. Virtual memory systems therefore operate at the unit of a *page* when transferring data to and from memory and disk. A page does not necessarily correspond to a disk block size but is fixed for a particular MMU typically between 4 and 64 kilobytes. This will usually correspond to several disk blocks.

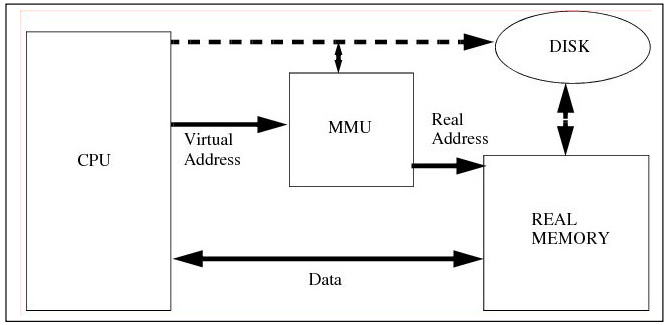


Figure 1 shows the basic structure of a paged virtual memory system.

The addresses issued by the CPU are virtual addresses which are the full width of the processor's addressing capability. In a modern 32 bit processor (for example ARM or x86) these will be a full 32 bits capable of addressing 4Gbytes of memory. The real memory will usually be smaller, typically 512 Mbytes on a laptop.

Each of these memory spaces is considered to be divided into pages. Note that this is not reflected in the internal structure of any memory, it is just an abstract division; the memory addresses are still binary values which cover a linear address space (usually at the unit of bytes). However, we can view any address as being divided into two sections, one which addresses the page and one which is an offset within the page which addresses individual units (bytes). By convention, we usually refer to the virtual address as having *virtual page numbers* (or just page numbers) and the real address as having *page frame numbers*. Figure 2 shows this pictorially.

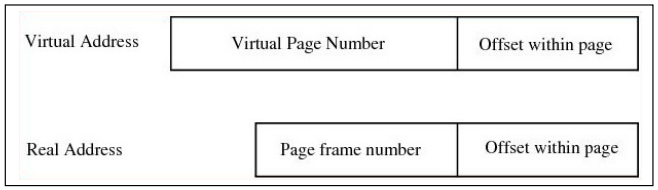


Figure 2 shows this pictorially view of *virtual page numbers* and *page frame numbers*.

The function of the MMU is to keep track of which virtual pages exist in real memory and, which on disk. It does this by keeping a *page table* of every possible virtual page number which contains the page frame numbers of any real copies. It is important to note that the offset is common to both virtual and real addresses. In order to find the data associated with a virtual address it is only necessary to take the virtual page number and replace it by the page frame number (if it exists). The page frame number is looked up in the page table to produce the real address which can then be used to access real memory. This is usually referred to as *address translation.*

If the table indicates that a virtual page does not exist in real memory this is called a *page fault.* It is necessary for the MMU to initiate a transfer of the page's data from the background storage (disk). The exact mechanism for this is not relevant here but it should be noted that this is a relatively slow process and the CPU itself can organise most of the transfer.

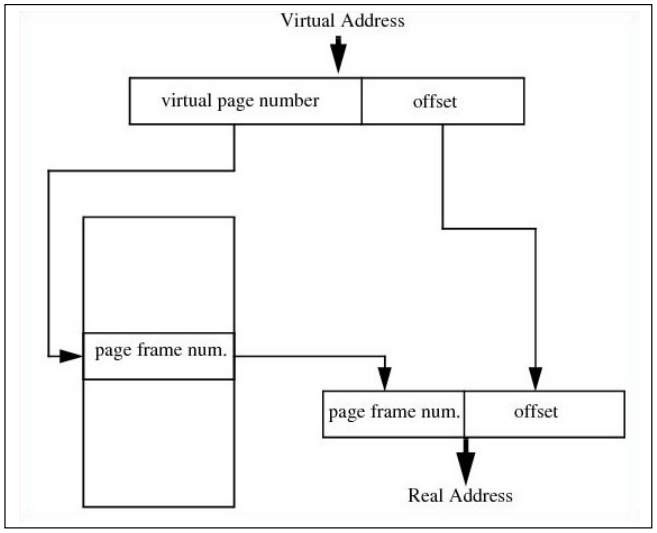


Figure 3 shows more detail of the MMU table and the address translation operation.

The page table is shown here as a single linear structure with the lookup done by the full page number. For large virtual address spaces, this may not be practical and more complex techniques may be required (this is covered in the course lectures).

4. Page Rejection

If there is a page fault, a place needs to be found in real memory for the page copy. Initially, assuming all real memory is not in use, it is possible to keep allocating page frames linearly through the address space. However, if all real memory is in use, we must find a page to reject. We must then write its contents back to disk before reallocating its page frame number to the page access, which has caused the fault.

We need to decide which page to choose using a *page rejection algorithm.* The most commonly used is an algorithm called LRU (least recently used). It works on the simple principle that the page, which has survived the longest time without being accessed, is a good candidate for replacement.
One useful optimisation is to note if a page has been written to since it was bought from disk into real memory. If not, there is no need to write it back to disk before reusing the real page frame. This is often true for pages which contain code.

5. The Exercise

The exercise is composed of two parts:

Part One: implements the majority of the code and the LRU page replacement algorithm. The majority of the (theoretical) advice below will aid you in this task;

after you have designed, developed, and implemented the program utilise the two trace files: **traceA** and **traceB**; to test your program.

Part Two: implements the a second algorithm the FIFO page replacement algorithm. The advice below will not aid you in this task. Nevertheless, it is expected that once you have implemented (and tested) the LRU algorithm you will have gained sufficient knowledge to enable you to undertake the FIFO design, development and testing on your own. After you have designed, developed, and implemented the program utilise the trace file: **traceLRUandFIFO**; to test your program. The FIFO algorithm has been covered in your lecture series and it is covered extensively in your course books; also see the Learning Resources (book & Web resources) as well as the Background Reading (book & Web resources) at the end of Lecture COMP25111 Operating Systems Lectures 14: Virtual Memory (3) for further reference material.

Introduction to Exercise

The purpose of the exercise is to write Java code to implement a simple MMU, which contains a page table and code to perform the necessary actions when read, or write accesses occur to a virtual memory address. Because the exercise is using a trace of addresses rather than executing a real program, there is no need to perform any real memory accesses. Instead, it is simply necessary to access the page table to determine if there is a page fault and, if so, find a free page frame to use and place it in the table. If, initially, there are unused page frames these can be allocated linearly until there are none free. At that point it will be necessary to find a page to reject and use its page frame for the newly required page.

You should implement a LRU algorithm to determine page rejection. To make this simple, you should have a 'timestamp' in each page table entry, which is updated appropriately, when a page is accessed.

You should also include a 'dirty' indication in the page table implementation, which is set on a write. This indicates if the page needs to be written back to disk if it is rejected.

Note that a practical page table will usually also contain page access information which determines the types of access which are allowed to a particular page (for example read only). There is no need to include such information for the purposes of this exercise.

In appropriate Blackboard subfolder, you will find a program harness which, after creating a MMU object, reads the trace of store accesses and makes calls to read and write methods within that object until the trace file ends. It then prints out some statistics:

Total Accesses
Total Instruction Fetches
Total Page Faults
Total Page Rejections (i.e. ignoring startup page faults)
Total Page Writebacks (to disk)

The first two are included in the given program; the last three need to be calculated by your MMU implementation. (A skeleton MMU.java file is provided in the appropriate Blackboard subfolder)

The trace file format is a line for each access. The first number indicates the access type (0=read, 1=write, 2=fetch) and the second is a (hex) 24 bit memory address. Within the program, the address is considered to be a 12-bit page number and a 12 bit offset representing 4k pages each of 4k bytes, i.e. 16Mbytes in total of virtual memory

space. This is deliberately small so that your implementation can use a simple single level page table. Although obviously not representative of modern systems, this is typical of a virtual memory system in computers of the 1960s when virtual memory was invented (in Manchester University on the Atlas computer [1][2]).

An example of 'traceA' file is given below:

```
2 400000
0 600000
2 400001
1 600001
2 400002
0 700000
2 500000
0 700001
```

The first line "2 400000" is decoded as follows:

First number	Second number
2	400000
2=fetch	0100 0000 0000 0000 0000 0000_{HEX}
This is a fetch access	24 bit hexadecimal memory address

The following decode the first number of each of the eight lines in traceA:

First number	Type of access
2	2=fetch
0	0=read
2	2=fetch
1	1=write
2	2=fetch
0	0=read
2	2=fetch
0	0=read

The program given makes a single call to the method which performs the trace simulation using a very very small real memory (2 [real] page frames, i.e. 8k bytes [in MMUSim.java the real memory sizes (in pages) is set to "rmem_pages = 2;"). This should be run with the file traceA when initially debugging your implementation. This file contains only 8 store accesses which have been devised so that you should be able to predict what statistics your MMU should produce.

When you think this is working, you should modify the program to perform simulations using both 32 and 64 real page frames on traceB. This represents 128k and 256k bytes of real memory, again realistic in the 1960s.

6. Results

You should be prepared to demonstrate the results of the runs on traceA and traceB when your exercise is marked. To help you determine if you have got a correct implementation, the total number of page faults for 32 pages and traceB should be 227.

You should also be prepared to describe your code and draw relevant conclusions from the results.

Assessment

COMP20051 Exercise 3: Developing Simulation of Paging Behaviour

Demonstrators please fill in the student's full detail [below] before starting the assessment; may be the best way to do this is to get the students to fill this in themselves:

Students full name	Student Number	Email address	Course studied

At the deadline, at the end of the Session, the marks awarded are as given in table 1.
Demonstrators please assign marks for each of the eight questions in boxes provided.

At the deadline, at the end of Session 5, the marks awarded are as follows:

Mark awarded for:	Exercise 3
1. Show Correct page table [data] structure; plus comment on how you [the student] developed the data structure.	1
Demonstrating an overall [software engineering] correct approach; of the algorithmic and combined data structure: **2** – **5**.	
2. State how REQUIREMENT development was undertaken.	1
3. State how [if any] ABSTRACT DESIGN was undertaken.	1
4. State how IMPLEMENTATION was undertaken.	1
5. Demonstrating a methodology for TESTING the program; and comment on: **interpretation of results**.	1
6. Showing the generation of an appropriate OUTPUT trace file: **traceA** files [displayed in real-time on command [Prompt] screen] (using LRU).	1
7. Showing the generation of an appropriate OUTPUT trace file: **traceB** files [displayed in real-time on command [Prompt] screen] (using LRU).	2
8. Showing the generation of an appropriate OUTPUT trace file: **traceLRUandFIFO** files [displayed in real-time on command [Prompt] screen] (using LRU and FIFO).	2
TOTAL mark	**10**

Remember to save these exercises in a directory nominally named …/COMP20051/ex3.

Do this exercise [or save this exercise] in a directory named COMP251111/ex3 directory, save your downloaded template [Skeleton(s)] code:-

MMUSim.java; and
MMU.java.

as well as the trace files:

traceA;
traceB; and
traceLRUandFIFO

… in COMP25111/ex3.

Remember to save your trace files of your solution.

The three files traceA; traceB; and traceLRUandFIFO; as well as the template [Skeleton(s)] MMUSim.java & MMU.java are downloadable from Blackboard in Folder Lab3.

References

[1] One-Level Storage System, T. Kilburn, D.B.G. Edwards, M.J. Lanigan, F.H. Sumner, IRE Trans. Electronic Computers April 1962

[2] Tom Kilburn (1956), The Atlas, School of Computer Science: information on Tom Kilburn's [computing] effort known as the MUSE (microsecond) computer , available [on-line] @ http://www.computer50.org/kgill/atlas/atlas.html, [Last accessed 12/10/09, 11:09].

7. Decoding the COMP20051 Laboratory Exercise 3; and '*supporting advice*'

The previous six pages document is a typical laboratory that you are normally set in the School of Computer Science. The first task is to decode [work out] what explicitly is required. As you have read the document once now reread section 5; this time highlighting or extracting the main requirements of the exercise.

Hint 1: [#H1]

[#H1] If you require a hint, download them from Blackboard. 'Hint No 1' hints at the requirements you should have data mined from reread section 5.

After reading document 'H1.doc' to validate you have extracted the relevant requirements you next step is to start the design phase that fulfils the requirements [#R1 to #R5] of the exercise. Then once designed the MMU class can be implemented.

Hint 2: [#H2]

[#H2] If you require a hint, download them from Blackboard. After you have compiled a list of requirements, you should now think about designing the MMU class. 'Hint No 2' suggests an approach you could take.

Reading document 'H2.doc' may help you move forward in the design phase or validate your design.

You should now be able to implement, test, and run MMUSim utilising traceA. Once you have reached this stage you can compare your program's trace output with a typical trace A output [result] available on Blackboard; the file is named 'traceAres.'

One of the final requirements of the exercise is to modify the program [MMUSim] to perform simulations using both 32 and 64 real page frames on traceB. To do this you should consider altering the number of real memory pages.

Finally, you should consider the best way to describe your code; when asked. In addition, you should think about drawing relevant conclusions from the results.

The ">>" operator

Final point: what does the operator ">>" actually do?

In the code snippet:

int addr = access.addr;

int vpage_no = addr >> 24 - vmem_bits; // assumes a 24 bit virt addr
*given vmem_bits = 12. Hence the actual value of '24 - vmem_bits' is "**addr >> 12**"*

In "doSimul(…)" in class MMUSim.Java. The operator ">>" is utilised. If we look at what happend

Values:
addr = 4194304
vpage_no = 1024

So what has happened?

The java operator "**>>**"

>> shift bits right with sign extension

It implies [with two objects '**x**' & '**y**']:

x >> y

Means: Shift Right - Signed

Explicitly means: Shift **x** to the right by **y** bits. Low order bits are lost.
Same bit value as sign (0 for positive numbers, 1 for negative) fills in the left bits.

Given the example addr = 4194304
which is specified for addr an interger (int) which is base (or radix) 10.

Converting 4194304_{10}
Converting to hexadecimal:
00400000_{16}
or radix 2 it is:
$0000\ 0000\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000_2$

Hence given "***addr >> 12***" After shifting right 12 places this is:

$0000\ 0000\ 0000\ 0000\ 0000\ 0100\ 0000\ 0000_2$
which is:
00000400_{16}
or radix 2 it is:
1024_{10}

QED