SRN- PES1201800110

STUDENT'S NAME	TEJAS SRINIVASAN	TOTAL MARKS OBTAINED
CLASS 4TH SEM 'A'	SUBJECT MPCA	
ROLL NO. -5'	DATE 24/05/2020	

UE18CS253 - MPCA Assignment [30 MARKS]

1 a) Differentiate between RISC and CISC architecture with two examples for each.

ANS.

	RISC	CISC
1.	It stands for "Reduced Instruction Set Computing"	It Stands for "Complex Instruction Set Computing".
2.	RISC processors have a smaller set of instructions with a few addressing modes.	CISC processors on the other hand have a large set of instructions with many addressing modes.
3.	Data Transfer is only permitted between memory and register	Data transfer is permitted between a) memory to register b) memory to memory c) register to register
4.	Emphasizes Compiler complexity	Emphasizes hardware complexity
5.	All instructions are executed in one cycle, which is equal to the largest execution time of an instruction.	It may take a variable number of clock cycles to complete a single instruction
6.	Supports pipelined execution	Does not support pipelining
Eg-	AMD 29k, ARM, MIPS	Eg- VAX, Intel x86

Caliber

1. b) Write the assembly equivalent code for the following code:

```
if (A==B):
        A = B+10
ELSE:
        A = A-10
```

Where A and B are memory locations having some integer values.

ANS. Assembly eq code:-

```
        LDR  R0, =A
        LDR  R1, =B
        CMP  A, B
        BEQ  L1
        SUB  A, A, #10
        SWI  0x011
L1:
        ADD  A, B, #10
        SWI  0x011
```

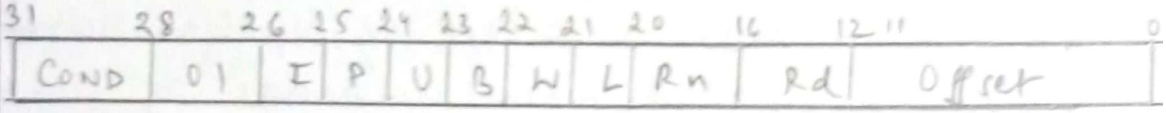
* Additionally we can specify the content in the memory locations of A and B if we wish to,

```
A:  .WORD   10, 20, 30, 40
B:  .WORD   1, 2, 3, 4
```

ROLL NO. | DATE

1. c) Encode the following instructions:

a. LDR R0, [R1, #4]



Opcode (Cond) = 1110 (always)
I = 0 (Taken as 0 because immediate value is present)
P = 1 (pre index / auto index) L = 1 (load)

So we get,

1110 | 01 | 0 | 1 | 1 | 0 | 0 | 1 | 0001 | 0000 |
0000 0000 0100 → 04H

→ Grouping these into 4 bits each, we get -

E 5 9 1 0 0 0 4

which is the required encoded instruction.

b. STR R0, [R1], #4

Cond = 1110 (always) I = 0 P = 0 (post index)
L = 0 (store instruction)

So we get,

1110 | 01 | 0 | 0 | 1 | 0 | 0 | 0 | 0001 | 0000 |
0000 0000 0100

→ Grouping these into 4 bits each we get -

E 4 8 1 0 0 0 4

which is the required encoded instruction.

Scanned with CamScanner

2.a) What is a control hazard? Explain with an example

ANS: A control hazard is a type of pipeline hazard that is incurred when a change of flow instructions changes the flow of instruction addresses, making it 'not sequential'.

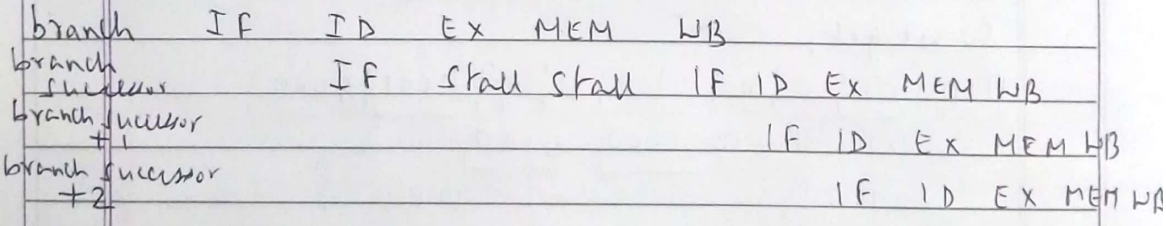
Types of instructions that can lead to a control hazard are -

1. Conditional branches (beq, bne)
2. Unconditional branches (b, bal)
3. Procedural calls (stack instructions like STRDB, LDMDA)
4. Proceduralreturns (MOV PC, LR)
5. Exceptions

Eg For the assembly instructions -

add R0, R1, R2
beq R0, R1, (L1)
and R4, R5, R6

* The five stage pipeline diagram for this is shown below -



* Since the branch instruction is in decode stage when the add instruction is in the execute stage, the branch cannot read the output until it is written to register file. To deal with this we can include two stall cycles but this will lead to the wastage of clock cycles and if the branch condition turns out to be true, it will render the Fetch redundant and hence waste 3 clock cycles totally.

STUDENT'S NAME		TOTAL MARKS OBTAINED
CLASS	SUBJECT	
ROLL NO.	DATE	

2-b) Can data hazard be eliminated with data forwarding? Under what situation it cannot be eliminated? Explain with an example.

Ans. Data hazards occur when the pipeline changes the order of read/write accesses to operands so that the order differs from the order seen by sequentially executing instructions on an unpipelined machine.

* Data forwarding is a simple hardware technique that simply forwards the output of the ALU operation to the end of Ex stage, for the instructions that require it. Since, the data is available at the execution stage, we do not have to wait for the write back (WB) stage and the data can be sent to the next instruction. This will help eliminating the stalls for the hazard.

Unfortunately, not all potential data hazards can be eliminate by the data forwarding technique.

Consider the following sequence of instructions -

```
LW   R1, 0(R1)
SUB  R4, R1, R5
AND  R6, R1, R7
OR   R8, R1, R9
```

IF ID EX MEM WB
IF ID EX_{SUB} MEM WB
IF ID EX$_{AND}$ MEM WB
IF ID EX MEM WB

* The problem with this sequence is that the load (LW) operation will not have data at the end of the MEM stage, while the SUB instruction needs to have this data at the beginning of the EX stage.

* The load instruction can forward the data (results of the AL operation) to the AND in EX stage and the OR instruction in ID stage but not to the SUB instruction as it would mean that the results would have to be forwarded in "negative time" which is not possible.

To deal with this problem, we must 'detect' the hazard and 'stall' the pipeline till the hazard is cleared.

IF ID EX MEM WB
IF ID Stall EX_{SUB} MEM WB
IF Stall ID EX MEM WB
Stall IF ID EX MEM WB

(Stalls have been inserted to ensure legal forwarding)

→ The only necessary forwarding is done from MEM to EX_{SUB} for R1. There is no need to forward data for the AND instruction now as it receives it in ID stage.

2c) Consider the following RISC assembly code

```
Load r1, 45(r2)      (1)
add r7 ← r1, r5      (2)
Sub r8 ← r1, r6      (3)
or  r9 ← r5, r1      (4)
brneq r7, target     (5)
add r10 ← r8, r5     (6)
xor r2 ← r3, r4      (7)
```

Identify each dependence; list the two instructions involved; identify which instruction is dependent; and if there is one, name the storage location involved (register or memory).

ANS > Instruction (1) is the load instruction and it has no data dependency.

> Instructions (2), (3) and (4) are dependent on the load Instruction (1) through the contents of the register r1

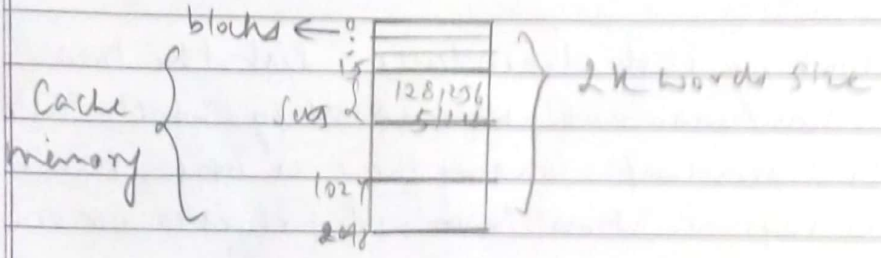
> Instruction (5) is dependent on Instruction (2) through r7

> Instruction (6) is dependent on Instruction (3) through r8.

> Instructions (6) and (7) are also dependent on the control statement (branch - brneq) since they are executed after that control logic is executed.

3. a) Explain with example and justify your answer why fully Associative mapping Cache is faster than direct mapping. Explain with example and ~~suggest~~ justify your answer why Set Associative mapping is faster than Direct Mapping.

Ans. Assume, a cache containing 128 blocks of 16 words each as shown below.
The total size of the cache is 2k words



In the direct mapped technique, block j of the main memory would map to block j modulo 128 in the cache, due to which there is more than one memory being mapped to the same storage location, which leads to block contention

> However, in fully associative mapping, each block can be placed anywhere in the cache and hence there is much less potential for a collision and it more flexible to search for an element as well. (12 tag bits, 4 identify the word in a block)

> In a set-associative mapping, each group of blocks is called a 'set'. The checking for a cache hit within the set can be done in parallel and hence even though the direct mapping technique has a shorter critical path, the set associative mapping is still more flexible & easy to search

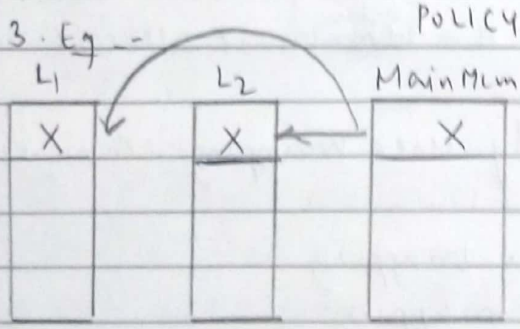
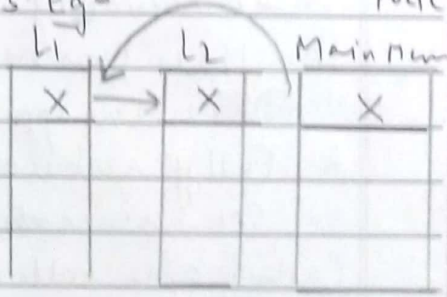
(4-Word
6-Tag
6-Set)

* Due to these various reasons, both associative and set-associative mapping are faster than direct mapping.

STUDENT'S NAME		TOTAL MARKS OBTAINED
CLASS	SUBJECT	
ROLL NO.	DATE	

3.b) Differentiate between the need for use of Inclusive cache versus the need for use of exclusive cache

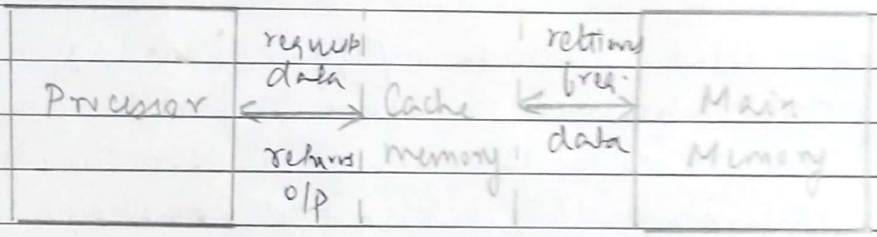
ANS

Inclusive Cache	Exclusive Cache
1. An inclusive cache is one where the same data can be present in both the L1 and L2 caches	1. An exclusive cache is one in which the data can be found in only one of the caches and the address of the instruction cannot be found in L1 and L2 at the same time.
2. Inclusive caches simplify cache coherence, however their trade off is lower performance (if the size of the LLC - 'largest level cache' is not significantly larger than the sum of all smaller caches.	2. Exclusive caches on the other hand can store more data and are more efficient in performance but they have the disadvantage of a complex cache coherence policy.
3. Eg - INCLUSIVE POLICY - 	3. Eg - EXCLUSIVE POLICY - 
If a value X is purged, it is only removed from L1 and not L2	If a value X is purged it is removed from L1 and passed to L2.

3. c) Write a note on cache memory.

ANS * Cache memory is a special type of high-speed memory. It is used to speed up and synchronize the main memory and the processor. It holds frequently requested data and instructions so that they are immediately available to the processor when required.

* The main principle behind the cache memory is called "locality of reference", which means that the many instructions in localized areas of the program are executed much more frequently.



* When the processor needs to read/write a location in the main memory, it checks for a corresponding entry in the cache. If the memory location is present in the cache, it is a "cache hit", else it is a "cache miss" in which case the Cache must allocate data from main memory.

The three commonly used mapping techniques are -
1. Direct mapping
2. Fully associative mapping
3. Set associative mapping.

Caches also follow a hierarchy -
1. L1 - primary or internal cache
2. L2 - secondary cache, slower than L1
3. L3 - tertiary cache, slower than L1 & L2.

STUDENT'S NAME		TOTAL MARKS OBTAINED
CLASS	SUBJECT	
ROLL NO.	DATE	

4.a) Consider the following code sequence

```
STR  R3, 512(R0)
LDR  R1, 1024(R0)
LDR  R2, 512(R0)
```

Assume direct mapped, write through cache that maps 512 and 1024 to the same block. Four words write buffer that is not checked on a read miss. Will the value in R2 always be equal to the value in R3? Explain with steps.

ANS:
* This is a Read-after-Write (RAW) hazard in memory.
* The data in R3 is placed into the write buffer after store instruction
* The following load instruction uses the same cache index and is therefore a miss
* The second load instruction tries to put the value in location 512 into R2, which also results in a miss.
* If the write buffer hasn't completed writing to location 512 in memory, the read operation of location 512 will put the old, incorrect value into the cache block, and subsequently into R2.

So, if proper measures are not taken to correct this hazard, the value of R2 will not be equal to that of R3.

Caliber

4(a). Explain with examples different ILP techniques

ANS: Instruction level Parallelism (ILP) is a measure of how many operations in a computer program can be performed in "parallel" at the same time.

→ This allows the compiler and processor to overlap the execution of multiple instructions, or even change the order of execution.

Some micro-architectural techniques that use ILP include -

① Superscalar processors/pipelines -
These are processors that issue more than one instruction per clock cycle. Each instruction processes one data item, but there are multiple execution units within the CPU and these multiple instructions can be processing several data items concurrently. They can be broadly divided into -
- Parallel pipelines (same pipelined - functional units for each stage)
- Diversified Pipelines (have special functional units that operate simultaneously in each stage)

② Out of order execution (OOOE)
Here, instructions are executed in an order of availability of data operands, instead of the original order, which makes sure that the processor can avoid being idle till the next instruction is fetched, and hence efficiently uses instruction cycles.

STUDENT'S NAME		TOTAL MARKS OBTAINED
CLASS	SUBJECT	
ROLL NO.	DATE	

(1) The amount of parallelism in a basic block (a straight line code sequence without branches) is limited by the instruction dependency and size of the basic block.
Techniques that can be used to find parallelism and dispatch are -

* Static Scheduling
Exploiting parallelism amongst the iterations of a loop can increase parallelism amongst instructions. This can be accomplished by unrolling the loop which increases the size of the basic block present. We can also schedule the loop to reduce the number of stalls and hence the time taken.

Eg- Loop: LD F0, 0(R1)
ADD F4, F0, F2
S.D F4, 0(R1)
ADDI R1, R1, #8
BNE R1, xxx, Loop

(5 cycles / iteration)

UNROLLED →

Loop: L.D F0, 0(R1)
L.D F6, 8(R1)
L.D F10, 16(R1)
L.D F14, 24(R1)
ADD F4, F0, F2
ADD F8, F6, F2
ADD F12, F10, F2
ADD F16, F14, F2
S.D F4, 0(R1)
S.D F8, 8(R1)
S.D F12, 16(R1)
S.D F16, 24(R1)
ADDI R1, R1, #32
BNE R1, xxx, Loop
(14 cycles) 4 iteration)

↓

3.5 cycles / iteration !

Caliber

* Dynamic scheduling
This technique is basically loop unrolling in hardware and can be used to improve the performance of instructions that needed to be executed out of order and/or in parallel

④ Register Renaming
It is a technique that can be used to avoid unnecessary serialization of a program's operations imposed by the reuse of certain registers. It is also used to enable out of order execution and removes name dependencies

Eg.

```
LDR R1, [R0]          LDR R1, [R0]
ADD R1, R1, R2   ⟶    ADD R1, R1, R2
LDR R1, [R3]          LDR R4, [R3]
SUB R1, R1, R2        SUB R4, R4, R2
```

(register renaming)

⑤ Speculative execution
It allows the execution of parts of instructions before being certain whether a certain execution should take place. A commonly used form is 'control flow speculation' where instructions past a branch are executed before the target is determined.

⑥ Branch prediction
It is used to avoid stalling till the control dependencies are resolved. It is commonly used along with speculative execution to implement ILP.

STUDENT'S NAME		TOTAL MARKS OBTAINED
CLASS	SUBJECT	
ROLL NO.	DATE	

4.a) Explain Amdahl's Law and Gustafson's law with suitable examples.

Ans * Amdahl's Law states that the maximum speedup possible in parallelizing an algorithm is limited by the serial portion of the code.

Mathematically, speedup is given by -

$$\text{Speedup} = T_S / T_N$$

$$= \frac{T_S}{[(1-F)T_S + (F/N)T_S]}$$

T_S - Time taken for serial execution

T_N - Total time taken $= \frac{1}{(1-F) + (F/N)}$

F - parallelization factor

* Gustafson's Law states that the proportion of computations that are sequential normally decreases as the problem size increases. This is an observed phenomena and not a theorem.

→ The parallel execution is fixed. As the problem size increases, the number of parallel processing units (N) increases.

Mathematically, Speedup is given by -

$$S_S(n) = \frac{S+np}{S+p} = n + (1-n)S$$

When $p = 1 - S$ S_S - scaled speedup factor (offers a much greater speedup when compared to Amdahl's law)

Eg - A comparison of speedup given by Amdahl's law & Gustafson's law.

If N = 10, S = 40%, F = 60%

$$S = \frac{1}{(0.4) + (0.6/10)} = \boxed{2.17}$$

$$S_S = 0.4 + (0.6 * 10) = \boxed{6.4}$$

Caliber

A much greater speedup factor.

5 a) Explain in detail how cache coherence in multiprocessor environment is taken care of. Briefly describe the different protocols.

ANS In multiprocessor systems, when many processes need a copy of the same memory block maintaining consistency amongst these copies raises the 'cache coherence problem'.

These can be taken care of by two categories of protocols -

1. Snoopy Protocols - All processors snoop the transaction requests, and respond appropriately.
2. Directory Based Protocols - A directory is used to keep a track of the data shared and the sharers.

A brief description of ~~some types of~~ Snoopy protocols have been listed below -

(a) MSI protocol

This is a basic cache coherence protocol. For MSI, each block can be in one of the following possible states (identified with each letter of the name)

- Modified.

The block has been modified in cache (i.e data in cache is inconsistent with memory). So, a cache with a block in 'M' state is responsible to write the block to backing store when it is evicted.

STUDENT'S NAME		TOTAL MARKS OBTAINED
CLASS	SUBJECT	
ROLL NO.	DATE	

- Shared
This block is not modified and is present in atleast one cache. The cache can evict the data before writing it to the backing store

- Invalid
The block is invalid and must be fetched from memory or another cache if it is to be stored in this cache.

② MOSI protocol
This protocol is an extension of the MSI protocol It adds the following state to MSI protocol -

- Owned
It indicates that the present processor owns this block and will service requests from other processors for the block.

③ MESI protocol
This is the most widely used cache protocol The states that are different are -

- Modified
The cache line is present only in the current cache and is 'dirty' if the value is different from main memory. The cache is required to write data back before permitting any other read/write operations.

- Exclusive
This indicates that the cache line is present in the current cache only and is 'clean' (value present in main memory)

* There also exists a protocol called 'MOESI' that uses a combination of all the above mentioned states to achieve superior performance.

5b) Discuss:
a) data parallelism
b) task parallelism
c) function parallelism

Ans: a) data parallelism
It is the concurrent execution of the same task on each multiple computing core. Synchronous computation is performed and since there is only one execution thread operating on all sets of data, the speedup is more.

b) task parallelism
It is the concurrent execution of different tasks on the same or different data on each multiple computing core. Asynchronous computation is performed, and since each of the processors will execute in a different thread or process, the speedup obtained is less.

c) function parallelism
This is basically another name for 'task parallelism'. However, it dives into a more 'thread level implementation and thread level parallelism. Here, the application or task is split into different functional units that are processed parallely in such a way that the outputs are produced by a shared input on multiple threads.

STUDENT'S NAME		TOTAL MARKS OBTAINED
CLASS	SUBJECT	
ROLL NO.	DATE	

5c). Explain Out of Order Execution with a relevant example.

ANS Out of order execution (OOOE) is an approach used by high performance microprocessors to efficiently use instruction cycles and reduce the delay by changing the order of execution of the instructions.

Instructions are executed in an order of availability of data, rather than the original order, which ensures that the processor can avoid being idle till the next instruction is fetched.

Eg:- OOO Load / Store execution

Consider a sequence of instructions as shown below:-

```
LDR  R3, 0(R6)
ADD  R7, R3, R4
STR  R4, 0(R7)
SUB  R1, R2, R2
LDR  R8, 0(R1)
```

Let us assume that the next instruction to be dispatched at R6 is a cache miss. Then the load instruction cannot be completed and the subsequent ADD & STR instructions cannot be completed as well. However, the OOO processor schedules the instructions whose values are known 'cache hits' and executes them first, out of order. Hence the Load & Sub instructions can be executed first, followed by the others which will avoid wastage of bandwidth in waiting for the load instruction to have a 'cache hit'.