CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

148

Internal Assesment Test - II

Sub:	ADVANCED COMPUTER ARCHITECHTURE							Code:	10CS74
Date:	03/11/2016	Duration:	90 mins	Max Marks:	50	Sem:	VII-A,B	Branch:	CSE

Answer Any FIVE FULL Questions

		Marks	OBE	
			CO	RBT
1	Describe the basic structure of a Tomasulo based MIPS floating point unit. Also explain seven fields of reservation station.	[10]	CO3	L1
2 (a)	List the steps to unroll the code and schedule it.	[04]	CO3	L1
(b)	What is the drawback of 1-bit dynamic branch prediction method? Clearly state, How to overcome in 2-bit prediction. Give the state transition diagram of 2-bit predictor.	[06]	CO3	L2
3	Explain hardware based speculation in detail.	[10]	CO3	L2
4	What is correlating predictor? Explain with one example. What is tournament predictor?	[10]	CO3	L2
5 (a)	Explain in detail, Branch Target Buffers with neat diagram	[06]	CO3	L2
(b)	List advantages and disadvantages of loop unrolling.	[04]	CO3	L1
6 (a)	To achieve a speedup of 80 with 100 processors what fraction of original computation can be sequential?	[04]	CO1	L3
(b)	Explain the basic schemes of enforcing coherence in a shared memory multiprocessor system.	[06]	CO4	L2
7	With a neat diagram explain the basic structure of a centralized shared memory and distributed shared memory multiprocessor.	[10]	CO4	L2
8	Explain directory based cache coherence for a distributed memory multiprocessor system along with the state transition diagram.	[10]	CO4	L2

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1:	Describe the basic principles and trends in processor design-ILP, pipelining, temporal locality and spatial locality.	0	1	0	0	0	0	0	0	0	1	1	0
CO2:	Describe the Instruction Set Architecture of MIPS,8086.	0	0	0	0	0	0	0	0	0	1	0	0
CO3:	Explain various static and dynamic scheduling techniques, static and dynamic branch prediction.	0	0	0	0	0	0	0	0	0	1	1	0
CO4:	Explain basic organization of cache and virtual memories, pipelined processors, distributed and shared memory systems.	0	0	0	0	0	0	0	0	0	1	0	0
CO5:	Apply concept and principle of ILP, cache memory, virtual memory to design high-performance computer architecture.	3	0	2	2	0	1	0	0	0	0	2	0
CO6:	Analyze the performance of processor, shared memory and distributed memory architecture.	0	1	0	0	0	0	0	0	0	0	0	0

Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PO1 - *Engineering knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*; PO4 - *Conduct investigations of complex problems*; PO5 - *Modern tool usage*; PO6 - *The Engineer and society*; PO7- *Environment and sustainability*; PO8 – *Ethics*; PO9 - *Individual and team work*; PO10 - *Communication*; PO11 - *Project management and finance*; PO12 - *Life-long learning*

Advanced Computer Architecture Scheme and Solution

Q.1 A Description of MIPS floating-point unit (4 Marks)

Description of seven fields of reservation station (6 Marks)

Q.2 A Listing the steps to unroll code (4 Marks)

B Explanation of Dynamic branch-prediction(2 Marks)

Drawback of 1-bit prediction (1 Mark)

State transition diagram and explanation on 2-bit prediction(3 Marks)

Q.3 Description on Hardware based speculation, its execution steps, ROB (10 marks)
and floating-point unit.

Q.4 Explanation on correlating predictor (4 Marks)

Example(2 Marks)

Explanation on tournament predictor (4 Marks)

Q.5 A Explanation on BTB (5 Marks)

Diagram of BTB (1 Mark)

B Listing four pros and cons of loop unrolling (4 Marks).

Q.6 A Solving problem correctly with formula (4 Marks)

B Explanation on migration and replication (2 Marks)

Explanation on directory and snooping based protocol (4 Marks)

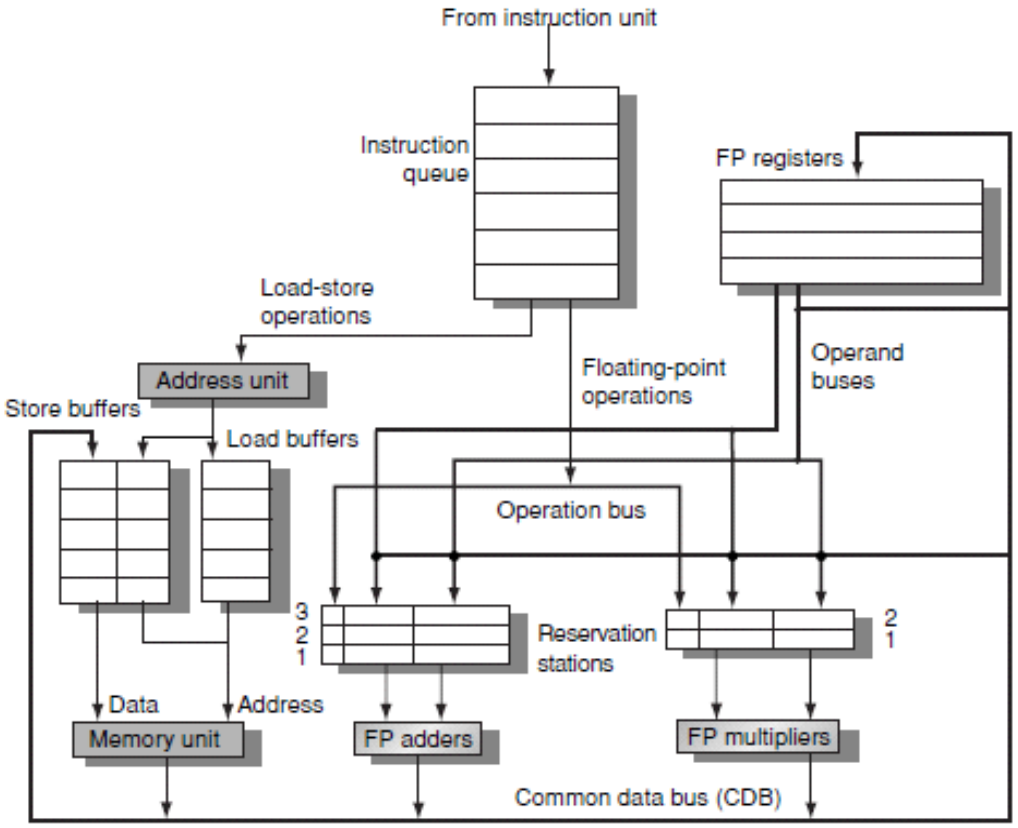
Q.7 Description on centralized shared memory multiprocessor (5 Marks)

Description on distributed shared memory multiprocessor (5 Marks)

Q.8 Explanation on directory-based protocol along with two state (10 marks)
transition diagrams one for cache and other for directory

Solution

1. The basic structure of Tomasulo based Floating point unit is shown below in diagram.



•Instructions are sent from the instruction unit into the instruction queue from which they are issued in FIFO order.

•Each reservation station holds an instruction that has been issued and is awaiting execution at a functional unit. The reservation station hold the operand values if they are already computed else it contains the names of the reservation stations that will provide the operand values.

•Load buffers have three functions: hold the components of the effective address until it is computed, track outstanding loads that are waiting on the memory, and hold the results of completed loads that are waiting for the CDB.

•Similarly, store buffers have three functions: hold the components of the effective address until it is computed, hold the destination memory addresses of outstanding stores that are waiting for the data value to store, and hold the address and value to store until the memory

unit is available.

- All results from either the FP units or the load unit are put on the CDB, which goes to the FP register file as well as to the reservation stations and store buffers.
- The FP adders implement addition and subtraction, and the FP multipliers do multiplication and division.

Seven fields of reservation station are as follows

- Op: The operation to perform on source operands S1 and S2.
- Qj, Qk—It is set when operand values are unavailable. It contains the reservation stations that will produce the corresponding source operand; a value of zero indicates that the source operand is already available in Vj or Vk, or is unnecessary.
- Vj, Vk— It is set when operand values are available. It contains the value of source operands. For loads, the Vk field is used to hold the offset field.
- A—Used to hold information for the memory address calculation for a load or store. Initially, the immediate field of the instruction is stored here; after the address calculation, the effective address is stored here.
- Busy—indicates that this reservation station and its accompanying functional unit are occupied.

2. A Steps to unroll the code and schedule it.

- Identify the loop body and loop termination code.
- Unrolling replicates loop body multiple times, adjusting the loop termination code.
- If we simply replicated the instructions when we unrolled the loop, the resulting use of the same registers could prevent us from effectively scheduling the loop. Thus, we will want to use different registers for each iteration, increasing the required number of registers.
- To perform scheduling a dependent instruction is separated from the source instruction by a distance in clock cycles equal to the pipeline latency of that source instruction. A compiler's ability to perform this scheduling depends both on the amount of ILP available in the program and on the latencies of the functional units in the pipeline.

Consider following code

```
Loop:  L.D      F0, 0(R1); F0=array element
       ADD.D    F4, F0, F2; add scalar in F2
```

```
        S.D       F4, 0(R1); store result
        DADDUI    R1, R1, #-8; decrement pointer;8 bytes (per DW)
        BNE       R1, R2, Loop; branch R1!=R2
```

•Without scheduling, every operation in the unrolled loop is followed by a dependent operation and thus will cause a stall. This loop will run in 27 clock cycles—each LD has 1 stall, each ADDD 2, the DADDUI 1, plus 14 instruction issue cycles—or 6.75 clock cycles for each of the four elements, but it can be scheduled to improve performance significantly. Loop unrolling is normally done early in the compilation process, so that redundant computations can be exposed and eliminated by the optimizer. Consider the loop is unrolled for 4 iterations. The unrolled loop is shown below.

```
Loop:   L.D       F0,0(R1)
        ADD.D     F4,F0,F2
        S.D       F4,0(R1) ;drop DADDUI & BNE
        L.D       F6,-8(R1)
        ADD.D     F8,F6,F2
        S.D       F8,-8(R1) ;drop DADDUI & BNE
        L.D       F10,-16(R1)
        ADD.D     F12,F10,F2
        S.D       F12,-16(R1) ;drop DADDUI & BNE
        L.D       F14,-24(R1)
        ADD.D     F16,F14,F2
        S.D       F16,-24(R1)
        DADDUI    R1,R1,#-32
        BNE       R1,R2,Loop
```

•After scheduling the loop is shown below and it takes only 14 clock cycles.

```
Loop:   L.D       F0,0(R1)
        L.D       F6,-8(R1)
        L.D       F10,-16(R1)
        L.D       F14,-24(R1)
        ADD.D     F4,F0,F2
```

```
ADD.D     F8,F6,F2
ADD.D     F12,F10,F2
ADD.D     F16,F14,F2
S.D       F4,0(R1)
S.D       F8,-8(R1)
DADDUI    R1, R1, #-32
S.D       F12, 16(R1)
S.D       F16, 8(R1)
BNE       R1, R2, Loop
```

2 A Dynamic Branch-prediction

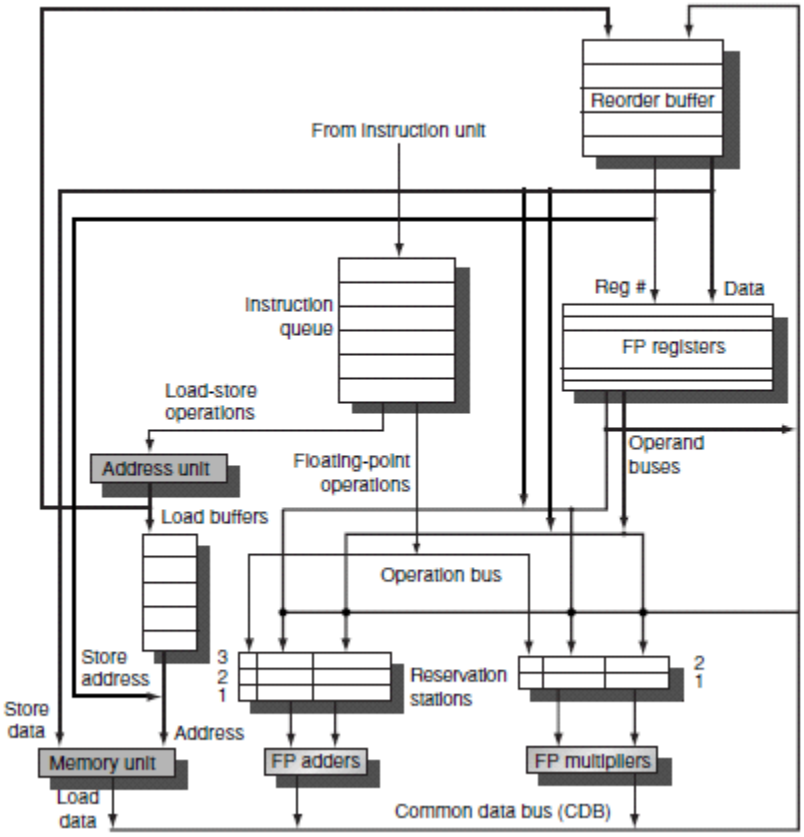
- The simplest dynamic branch-prediction scheme is a branch-prediction buffer or branch history table.
- The memory contains a bit that says whether the branch was recently taken or not.
- This simple 1-bit prediction scheme has a performance shortcoming: Even if a branch is almost always taken, we will likely predict incorrectly twice, rather than once, when it is not taken, since the misprediction causes the prediction bit to be flipped.
- To remedy this weakness, 2-bit prediction schemes are often used. In a 2-bit scheme, a prediction must miss twice before it is changed.
- Figure shows the finite-state processor for a 2-bit prediction scheme.
- By using 2 bits rather than 1, a branch that strongly favors taken or not taken—as many branches do—will be mispredicted less often than with a 1-bit predictor. The 2 bits are used to encode the four states in the system.
- The 2 bits are used to encode the four states in the system. The 2-bit scheme is actually a specialization of a more general scheme that has an n-bit saturating counter for each entry in the prediction buffer. With an n-bit counter, the counter can take on values between 0 and 2" - 1: When the counter is greater than or equal to one-half of its maximum value (2" - 1), the branch is predicted as taken; otherwise, it is predicted untaken.

3 Hardware based speculation

- Hardware-based speculation combines three key ideas: dynamic branch

prediction to choose which instructions to execute, speculation to allow the execution of instructions before the control dependences are resolved, and dynamic scheduling to deal with the scheduling of different combinations of basic blocks.

- In speculation, we fetch, issue, and *execute* instructions, as if our branch predictions were always correct. Also some mechanism is needed to handle the incorrect speculation.
- The key idea behind implementing speculation is to allow instructions to execute out of order but to force them to commit *in order* and to prevent any irrevocable action (such as updating state or taking an exception) until an instruction commits. Hence, when we add speculation, we need to separate the process of completing execution from instruction commit, since instructions may finish execution considerably before they are ready to commit.
- ROB is a source of operands for instructions, just as the reservation stations provide operands in Tomasulo's algorithm. The key difference is that in Tomasulo's algorithm, once an instruction writes its result, any subsequently issued instructions will find the result in the register file. With speculation, the register file is not updated until the instruction commits (and we know definitively that the instruction should execute); thus, the ROB supplies operands in the interval between completion of instruction execution and instruction commit.
- Each entry in the ROB contains four fields: the instruction type, the destination field, the value field, and the ready field. The instruction type field indicates whether the instruction is a branch (and has no destination result), a store (which has a memory address destination), or a register operation (ALU operation or load, which has register destinations). The destination field supplies the register number (for loads and ALU operations) or the memory address (for stores) where the instruction result should be written. The value field is used to hold the value of the instruction result until the instruction commits. The ready field indicates that the instruction has completed execution, and the value is ready.
- Figure below shows the basic structure of a FP unit using Tomasulo's algorithm extended to handle speculation. The major change is the addition of the ROB and the elimination of the store buffer, whose function is integrated into the ROB.



•Four steps involved in executing instruction

Issue stage

•Step 1: Instruction is issued from Instruction Queue.

•Step 2: Allocate a empty reservation station and empty ROB slot to instruction if available, otherwise instruction is stalled until it is available.

•Step 3: The ROB entry number of the issued instruction is sent to reservation station in order to tag the result when it is placed on CDB.

•Step 3: If operands available in registers or ROB , then buffer it in reservation station.

Execute stage

•If the operand is not available monitor the CDB until operand becomes available.

•When the operands are available execute the instruction.

Write Result

•After execution the result is written on CDB (with the ROB entry number that was sent when instruction was issued.)

•Also it is written into from CDB to ROB and any reservation station waiting for the result.

Commit stage

- This is final stage where instruction commits if speculation is correct and results are written in final register file or memory location.
- If the committing instruction is normal ALU or load instruction its result is updated in final register file and its entry is removed from ROB.
- If the committing instruction is normal store instruction its result is updated in memory and its entry is removed from ROB.
- If the committing instruction is branch and speculation is correct then execution is finished.
- If the committing instruction is branch and prediction is wrong i.e. incorrect speculation, then ROB is flushed and execution is restarted at the correct successor of the branch.

4. Correlating predictor

- Consider this MIPS code

```
DADDIU    R3,R1,      #–2
BNEZ              R3,L1       ;branch b1 (aa!=2)
DADD      R1,R0,R0    ;aa=0
L1: DADDIU R3,R2      ,#–2
BNEZ              R3,L2       ;branch b2 (bb!=2)
DADD      R2,R0,R0    ;bb=0
L2: DSUBU  R3, R1, R2  ;R3=aa-bb
BEQZ              R3, L3      ;branch b3 (aa==bb)
```

- Let's label these branches b1, b2, and b3. The key observation is that the behavior of branch b3 is correlated with the behavior of branches b1 and b2. Clearly, if branches b1 and b2 are both not taken (i.e., if the conditions both evaluate to true and aa and bb are both assigned 0), then b3 will be taken, since aa and bb are clearly equal. A predictor that uses only the behavior of a single branch to predict the outcome of that branch can never capture this behavior.
- Branch predictors that use the behavior of other branches to make a prediction are called *correlating predictors* or *two-level predictors*.

- Existing correlating predictors add information about the behavior of the most recent branches to decide how to predict a given branch.
- General form of correlating predictor is (m,n) predictor. A (*m,n*) predictor uses the behavior of the last *m* branches to choose from 2*m* branch predictors, each of which is an *n*-bit predictor for a single branch.
- The number of bits in an (*m*,*n*) predictor is

$2^{\wedge m} \times n \times$ Number of prediction entries selected by the branch address

Example How many bits are in the (0,2) branch predictor with 4K entries? How many entries are in a (2, 2) predictor with the same number of bits?

Answer The predictor with 4K entries has

$2^0 \times 2 \times 4K = 8K$ bits

How many branch-selected entries are in a (2,2) predictor that has a total of 8K bits in the prediction buffer?

We know that

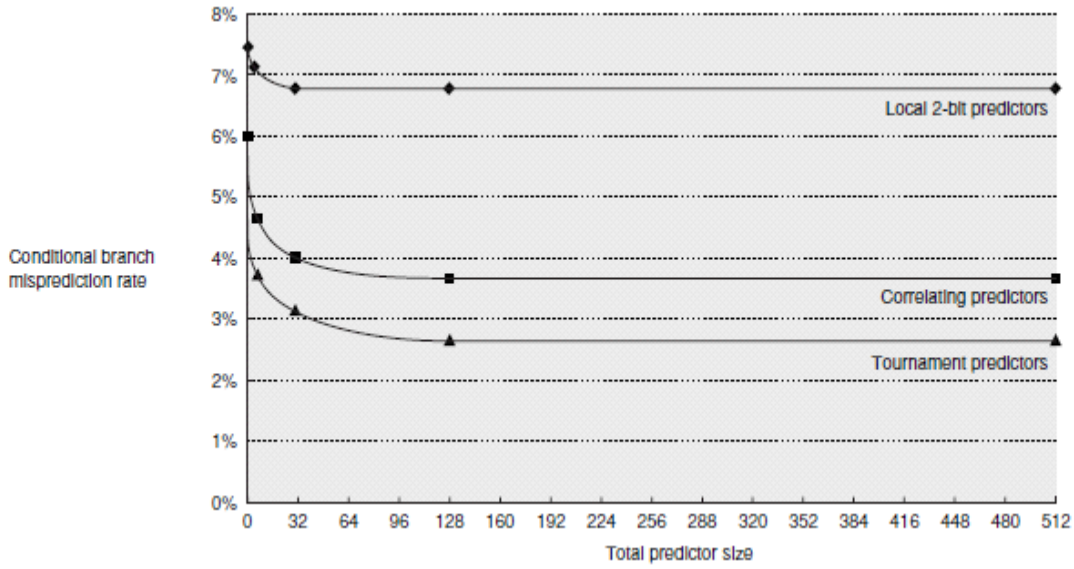
$2^2 \times 2 \times$ Number of prediction entries selected by the branch = 8K

Hence, the number of prediction entries selected by the branch = 1K.

Tournament Predictor

- Tournament predictor uses multiple predictors, some predictor use global information and some use local information.
- Tournament predictor is very accurate and makes use of very large number of prediction bits effectively.
- Every tournament predictor uses 2-bit saturating counter per branch which selects the most efficient predictor (local, global or even mix) for that branch.
- The advantage of tournament predictor is its ability to select right predictor for a particular branch.
- It is found that for spec integer programs global predictor is selected 40% of time and for SPEC floating point programs it is selected 15% of time.
- Figure below shows the performance of local 2-bit predictor, correlating predictor and

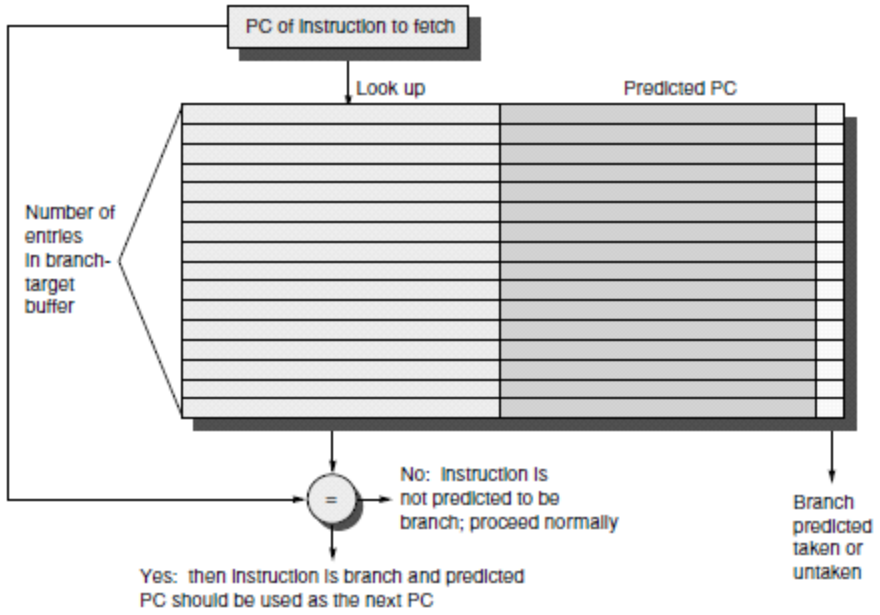
tournament predictor.



•Also Alpha 21264's tournament predictor uses 4K-2bit saturating counter to choose among global predictor or local predictor.

5. A Branch Target Buffer

•A branch-prediction cache that stores the predicted address for the next instruction after a branch is called a *branch-target buffer* or *branch-target cache.* Figure below shows a branch-target buffer.

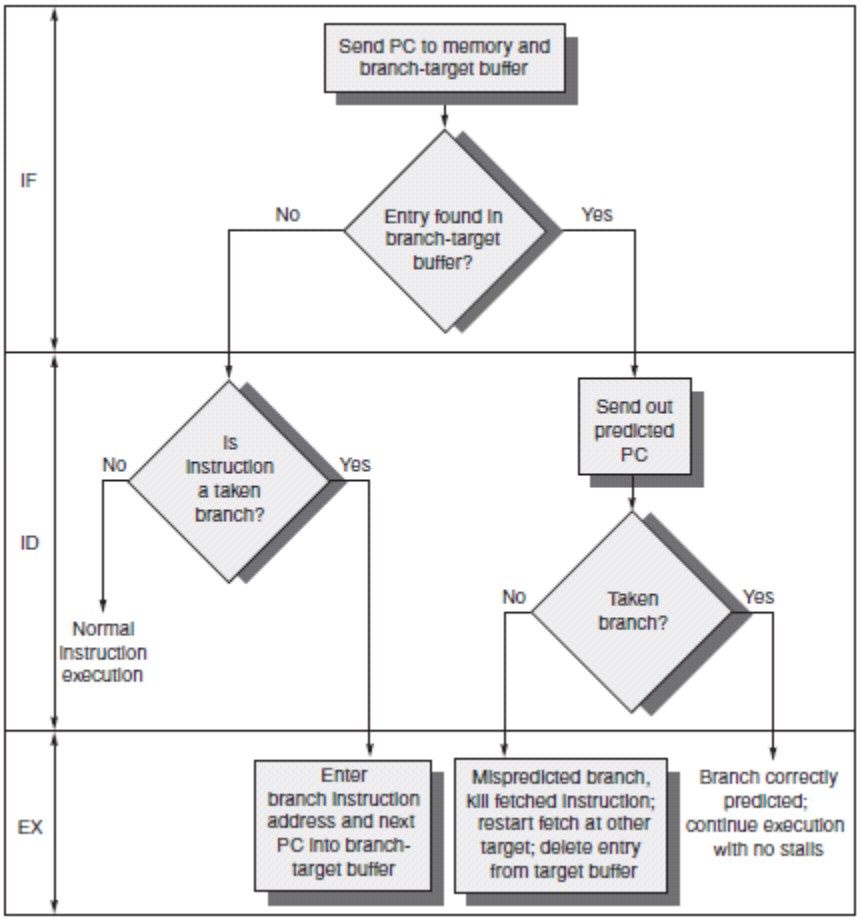


•Because a branch-target buffer predicts the next instruction address and will send it out *before* decoding the instruction, we *must* know whether the fetched instruction is predicted as a taken branch. If the PC of the fetched instruction matches a PC in the prediction buffer, then the corresponding predicted PC is used as the next PC. The hardware for this branch-target buffer is essentially identical to the hardware for a cache.

•The PC of the instruction being fetched is matched against a set of instruction addresses stored in the first column; these represent the addresses of known branches. If the PC matches one of these entries, then the instruction being fetched is a taken branch, and the second field, predicted PC, contains the prediction for the next PC after the branch. Fetching begins immediately at that address. The third field, which is optional, may be used for extra prediction state bits.

•We only need to store the predicted-taken branches in the branch-target buffer, since an untaken branch should simply fetch the next sequential instruction, as if it were not a branch.

•Figure below shows the detailed steps when using a branch-target buffer for a simple five-stage pipeline. From this we can see that there will be no branch delay if a branch-prediction entry is found in the buffer and the prediction is correct. Otherwise, there will be a penalty of at least 2 clock cycles.



5. B Advantages of loop unrolling

•Loop unrolling can also be used to improve scheduling.

•Determine that unrolling the loop would be useful by finding that the loop iterations were independent, except for the loop maintenance code.

•If the statements in the loop are independent of each other (i.e. where statements that occur earlier in the loop do not affect statements that follow them), the statements can potentially be executed in parallel

Disadvantage of loop unrolling

•Due to loop unrolling the code size grows.

•Also there is shortage of registers due to aggressive loop unrolling and scheduling.

6. A. To achieve a speedup of 80 with 100 processors what fraction of computation time is sequential?

Amdahl's Law is

$$\text{Speedup} = \frac{1}{\frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} + (1 - \text{Fraction}_{\text{enhanced}})}$$

$$80 = \frac{1}{\frac{\text{Fraction}_{\text{parallel}}}{100} + (1 - \text{Fraction}_{\text{parallel}})}$$

Simplifying this equation yields

$$\begin{aligned} 0.8 \times \text{Fraction}_{\text{parallel}} + 80 \times (1 - \text{Fraction}_{\text{parallel}}) &= 1 \\ 80 - 79.2 \times \text{Fraction}_{\text{parallel}} &= 1 \\ \text{Fraction}_{\text{parallel}} &= \frac{80 - 1}{79.2} \\ \text{Fraction}_{\text{parallel}} &= 0.9975 \end{aligned}$$

Thus, to achieve a speedup of 80 with 100 processors, only 0.25% of original Computation can be sequential.

6. B Basic schemes for enforcing coherence

- A program running on multiple processors will normally have copies of same data in several processors.
- In coherent multiprocessor the cache provides both migration and replication of shared data.
- Coherent caches support migration and data is migrated to local caches of all processors. It reduces the latency of access.
- It also provides replication of shared data item in caches of all processors. But this migration and replication is useful only for small scale multiprocessors.
- For large scale multiprocessors a protocol is used for maintaining coherence. There are two types of protocol.
- Directory based protocol: sharing status is maintained in directory. It has more overhead than snooping.
- Snooping protocol: Every cache has a copy of block of data, also has sharing status. The

caches are connected by broadcast medium i.e. bus or switch and all cache controllers monitor or snoop on medium to determine whether or not they have a copy of block that is requested on the bus or switch access.

•Here the snooping protocol is write invalidate protocol. The processor has a exclusive access for any item which it writes. Write invalidate protocol means any other processor which has same copy of data item are invalidated until the processor having exclusive access completes the writing operation.

•Figure shows an example of an invalidation protocol for a snooping bus with write-back caches.

Processor activity	Bus activity	Contents of CPU A's cache	Contents of CPU B's cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

•Since the write requires exclusive access, any copy held by the reading processor must be invalidated. Thus, when the read occurs, it misses in the cache and is forced to fetch a new copy of the data.

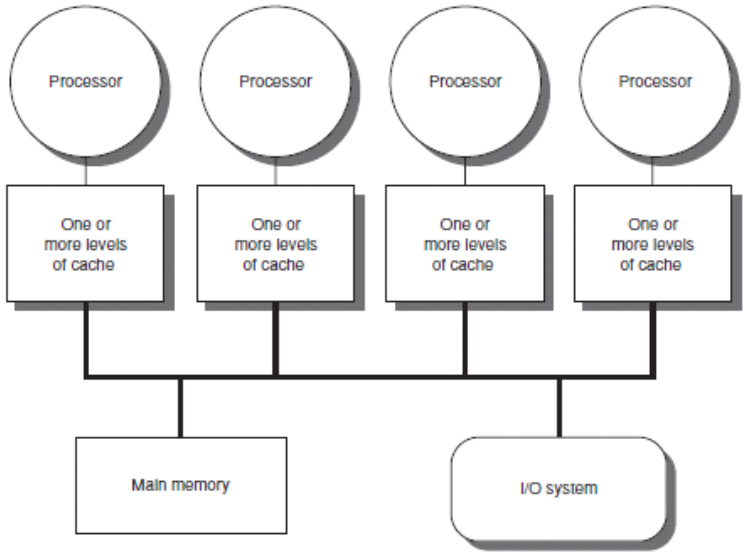
•If two processors do attempt to write the same data simultaneously, one of them wins the race (we'll see how we decide who wins shortly), causing the other processor's copy to be invalidated. For the other processor to complete its write, it must obtain a new copy of the data, which must now contain the updated value. Therefore, this protocol enforces write serialization.

•The alternative to an invalidate protocol is to update all the cached copies of a data item when that item is written. This type of protocol is called a *write update* or *writes broadcast* protocol. It consumes considerably more bandwidth.

7. **Two types of MIMD**

Centralized shared-memory architectures

- Multiple processor-cache subsystems share the same physical memory, typically connected by one or more buses or a switch. The key architectural property is the uniform access time to all of memory from all the processors.
- For multiprocessors with small processor counts, it is possible for the processors to share a single centralized memory.
- Scaling the number of processors is difficult in this type of architecture.
- Because there is a single main memory that has a symmetric relationship to all processors and a uniform access time from any processor, these multiprocessors are most often called symmetric (shared-memory) multiprocessors (SMPs), and this style of architecture is sometimes called uniform memory access (UMA), arising from the fact that all processors have a uniform latency from memory, even if the memory is organized into multiple banks.
- Figure below shows what these multiprocessors look like.

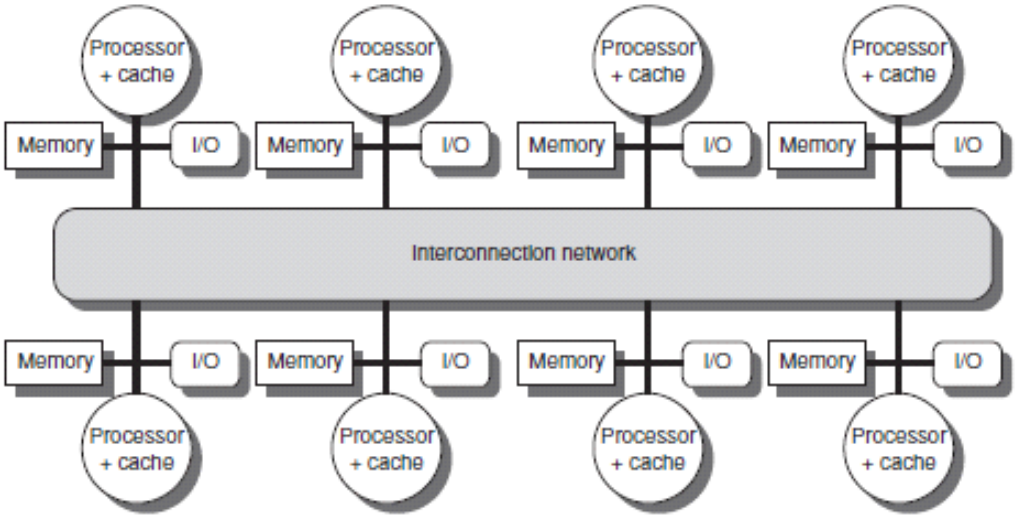


Distributed Shared Memory Multiprocessor

- The basic architecture of a distributed-memory multiprocessor consists of individual nodes containing a processor, some memory, typically some I/O, and an interface to an interconnection network that connects all the nodes.
- Distributing the memory among the nodes has two major benefits. First, it is a cost-effective way to scale the memory bandwidth if most of the accesses are to the local memory in the node. Second, it reduces the latency for accesses to the local memory. These

two advantages make distributed memory attractive at smaller processor counts as processors get ever faster and require more memory bandwidth and lower memory latency.

•The key disadvantages for distributed memory architecture are that communicating data between processors becomes somewhat more complex, and that it requires more effort in the software to take advantage of the increased memory bandwidth afforded by distributed memories.



8. Directory based protocol

•Just as with a snooping protocol, there are two primary operations that a directory protocol must implement: handling a read miss and handling a write to a shared, clean cache block.

•The states are as follows

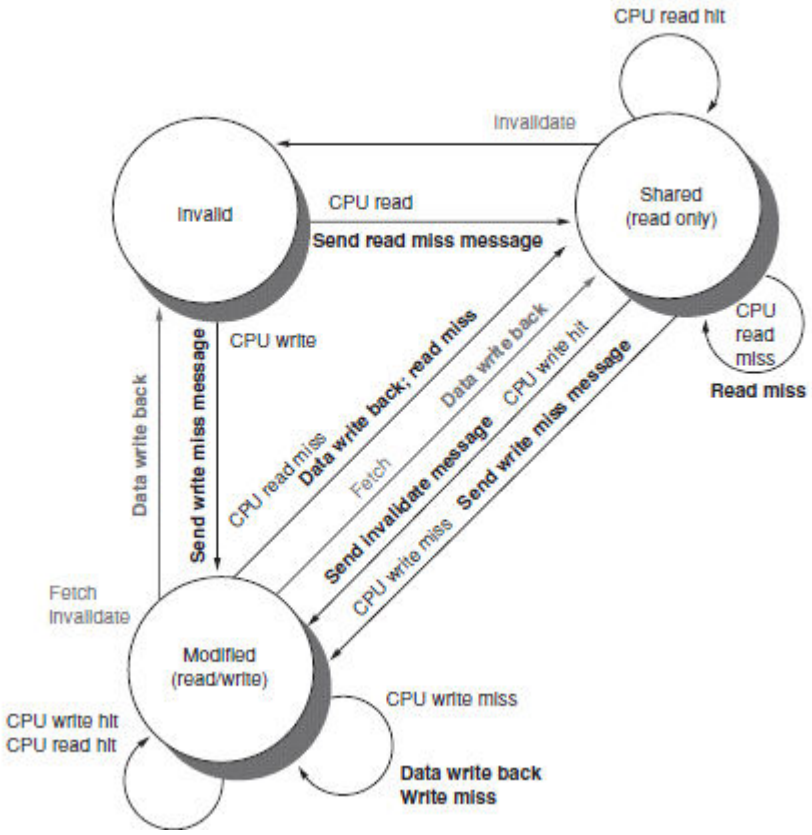
•*Shared*—one or more processors have the block cached, and the value in memory is up to date (as well as in all the caches).

•*Uncached*—No processor has a copy of the cache block.

•*Modified*—Exactly one processor has a copy of the cache block, and it has written the block, so the memory copy is out of date. The processor is called the *owner* of the block.

•The state transitions for an individual cache are caused by read misses, write

misses, invalidates, and data fetch requests; these operations are all shown in figure.



- The write miss operation, which was broadcast on the bus (or other network) in the snooping scheme, is replaced by the data fetch and invalidates operations that are selectively sent by the directory controller.
- Like the snooping protocol, any cache block must be in the exclusive state when it is written, and any shared block must be up to date in memory.
- In a directory-based protocol, the directory implements the other half of the coherence protocol. A message sent to a directory causes two different types of actions: updating the directory state and sending additional messages to satisfy the request.
- In addition to the state of each block, the directory must track the set of processors that have a copy of a block; we use a set called *Sharers* to perform this function.
- Figure below shows the actions taken at the directory in response to messages received.

The directory receives three different requests: read miss, write miss, and data write back.

•When a block is in the uncached state, the copy in memory is the current value, so the only possible requests for that block are

Read miss—The requesting processor is sent the requested data from memory, and the requester is made the only sharing node. The state of the block is made shared.

Write miss—The requesting processor is sent the value and becomes the sharing node. The block is made exclusive to indicate that the only valid copy is cached. Sharers indicates the identity of the owner.

•When the block is in the shared state, the memory value is up to date, so the same two requests can occur:

Read miss—The requesting processor is sent the requested data from memory, and the requesting processor is added to the sharing set.

Write miss—The requesting processor is sent the value. All processors in the set Sharers are sent invalidate messages, and the Sharers set is to contain the identity of the requesting processor. The state of the block is made exclusive.

•When the block is in the exclusive state, the current value of the block is held in the cache of the processor identified by the set Sharers (the owner), so there are three possible directory requests:

Read miss—The owner processor is sent a data fetch message, which causes the state of the block in the owner's cache to transition to shared and causes the owner to send the data to the directory, where it is written to memory and sent back to the requesting processor. The identity of the requesting processor is added to the set Sharers, which still contains the identity of the processor that was the owner (since it still has a readable copy).

Data write back—The owner processor is replacing the block and therefore must write it back. This write back makes the memory copy up to date (the home directory essentially becomes the owner), the block is now uncached, and the Sharers set is empty.

Write miss—The block has a new owner. A message is sent to the old owner, causing the cache to invalidate the block and send the value to the directory, from which it is sent to the requesting processor, which becomes the new owner. Sharers is set to the identity of the new owner, and the state of the block remains exclusive.