CS 704 Final Term Papers Solutions by:

Imran Ullah Gondal (ms150200602)

Long Questions:

Q.1.Find the die yield for a processor chip with the following manufacturing cost factors: die size = 380 mm^2, estimated defect rate = 0.75 per cm^2, α = 4 (5)

Solutions:

Die yield is the fraction or percentage of good dies on a wafer number

- Wafer yield accounts for completely bad wafers so need not be tested
- Wafer yield corresponds to on defect density by α which depends on number of masking levels
- Good estimate for CMOS is 4.0 and
- **Die yield =** Wafer yield x (1 + defects per unit area x die area) / α) $^{-\alpha}$
- Example: The yield of a die, 0.7cm on a side, with defect density of 0.6/cm^2

  = (1+[0.6x0.47]/4.0) $^{-4}$ = 0.75

Die yield = Wafer yield x (1 + defects per unit area x die area / α)) $^{-\alpha}$

Example: The yield of a die, 0.7cm on a side, with defect density of 0.75/cm^2

So,

= (1+[0.75 x 380]/4.0) $^{-4}$ = (1+285 / 4) $^{-4}$

(71.5) $^{-4}$ = $1/(71.5)^4$ = 0.00349

Q.2. If computer A runs a programs in 5seconds and computer B runs the same program in 10 seconds, how much slower is B than A? RR=3 (5)

Q.4. Briefly define the following terms: (10)

Pipelining:
A technique used in advanced microprocessors where the microprocessor begins executing a second instruction before the first has been completed. That is, several instructions are in the *pipeline* simultaneously, each at a different processing stage.
The pipeline is divided into segments and each segment can execute its operation concurrently with the other segments. When a segment completes an operation, it passes the result to the next segment in the pipeline and fetches the next operation from the preceding segment. The final results of each instruction emerge at the end of the pipeline in rapid succession.

Instruction Cycle:
An instruction cycle (sometimes called fetch-and-execute cycle, fetch-decode-execute cycle, or FDX) is the basic operation cycle of a computer. It is the process by which a computer retrieves a program instruction from its memory, determines what actions the instruction requires, and carries out those actions.

Finite State Machine: A finite-state machine (FSM) or finite-state automaton (plural: automata), or simply a state machine, is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of

Branch predication:
In computer architecture, a branch predictor is a digital circuit that tries to guess which way a branch (e.g. an if-then-else structure) will go before this is known for sure. The purpose of the branch predictor is to improve the flow in the instruction pipeline.

Dynamic Scheduling: Dynamic priority scheduling is a type of scheduling algorithm in which the priorities are calculated during the execution of the system. The goal of dynamic priority scheduling is to adapt to dynamically changing progress and form an optimal configuration in self-sustained manner.
A static schedule is some kind of list containing the order of the processes and the durations in which they are schedule

Q.6. How the operands and operations for media and signal processing differ from the normal integer operands and operation? Discuss (10)

Solution:
Operands:
Vertex – A common 3D data type dealt in graphics applications – four components: (x, y, z) and w=color or hidden surfaces – vertex values are usually 32-bit floating-point values – Three vertices specify a graphics primitive such as a triangle • Pixel – Typically 32 bits, consisting of four 8-bit channels • R (red), G (green), B (blue), and A (attribute: eg. transparency) • DSPs add fixed point – fractions between -1 and +1 (divide by 2n-1) • Blocked floating point – a block of variables with common exponent – accumulators, registers that are wider to guard against round-off error to aid accuracy in fixed-point arithmetic
Operation: Data for multimedia operations is often narrower than the 64-bit data word – normally in single precision, not double precision • Single-instruction multiple-data (SIMD) or vector instructions – A partitioned add operation on 16-bit data with a 64-bit ALU would perform four 16-bit adds in a single clock cycle • Hardware cost: prevent carries between the four 16-bit partitions of the ALU – Two 32-bit floating-point operations (paired single operations) • The two partitions must be insulated to prevent operations on one half from affecting the other
Instruction Cycle:
An instruction cycle (sometimes called fetch-and-execute cycle, fetch-decode-execute cycle, or FDX) is the basic operation cycle of a computer. It is the process by which a computer retrieves a program instruction from its memory, determines what actions the instruction requires, and carries out those actions.
Finite State Machine: A finite-state machine (FSM) or finite-state automaton (plural: automata), or simply a state machine, is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of

Q. 7. Consider a branch-target buffer that has penalties of 0,2 and 2 clock cycles for correct conditional branch predictions in correct prediction and a buffer miss, respectively. Consider a brand-target buffer design that distinguishes conditional and unconditional branches storing the target addresses for a conditional branch and the target instructions for an unconditional branch. (15)

Q.8. (a) Compare the Interrupts and Traps. Also differentiate the hardware and software interrupts by providing suitable examples. (10)

A trap is an exception in a user process. It's caused by division by zero or invalid memory access. It's also the usual way to invoke a kernel routine (a system call) because those run with a higher priority than user code. Handling is synchronous (so the user code is suspended and continues afterwards). In a sense they are "active" - most of the time, the code expects the trap to happen and relies on this fact.

An interrupt is something generated by the hardware (devices like the hard disk, graphics card, I/O ports, etc). These are asynchronous (i.e. they don't happen at predictable places in the user code) or "passive" since the interrupt handler has to wait for them to happen eventually.
You can also see a trap as a kind of CPU-internal interrupt since the handler for trap handler looks like an interrupt handler (registers and stack pointers are saved, there is a context switch, execution can resume in some cases where it left off).

b)How FSM control design approach differs from the Micro program controller? Give the Micro program controller to handle branch and jump instructions. (5)

Organization refers to how the features of a computer are implemented; i.e., control signal generation as FSM or microprogramming, memory technology-SRAM, DRAM etc, hardware or software based realization of operation-multiplication by hardware or algorithmically. Moreover, the organization of same architecture may differ between different versions; i.e., different versions of Intel x86 family may have different organizations

Finite State Machine: A finite-state machine (FSM) or finite-state automaton (plural: automata), or simply a state machine, is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of

Microprogramming

• Specialize state-diagrams easily captured by micro sequencer
  simple increment & —branch‖ fields
  datapath control fields
• Control design reduces to Microprogramming
• Microprogramming is a fundamental concept
  implement an instruction set by building a very simple processor and interpreting theinstructions
  essential for very complex instructions and when few register transfers are possible
  overkill when ISA matches datapath 1-1

Q.9 Comparing performance of two designs: the ratio, η = Execution time Y / Execution time X

determines how much lower execution time machine Y takes as compared to X ; as performance is inverse of execution time, i.e.,

η = Performance X / Performance Y

Concept of Local Decoding:

The local decoding concept is where instead of asking the Main Control to generates the ALUctr signals directly ; the main control will generate a set of signals called ALUop.

For all I and J type instructions, ALUop will tell the ALU Control exactly what the ALU needs to do (Add, Subtract, ...) .

2Bit Branch Pridction:

2 bits are used to encode 4-states in the system (counter) Say:

States 00 and 01 for Predict Not-Taken
States 10 and 11 for Predict Taken

In a saturating counter implementation:

2-bit counter saturates at:
- 00 (Predict Taken) or
- 11 (Predict Not taken)

The counter is incremented when a branch is taken and decremented when it is not taken; e.g.,

- 00 to 01 for Taken when predicted not taken
-10 to 11 for Taken when predicted taken

Q. 11:The 16-bit Zilog Z8001 has the following general instruction format:

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Mode Opcode w/b Operand2 Operand1

The mode field specifies how to locate the operands from the operand fields. The w/b filed is used in certain instructions to specify whether the operands are byte or 16-bit words. The operand 1 field may (depending on the mode field contents) specify one of 16 general-purpose registers. The operand 2 field may specify any general-purpose registers except register 0. When the operand 2 field is all zeros, each of the original opcodes takes on a new meaning.

(a) How many opcodes are provided on the Z8001?

Solution:

Let us analyze the problem a bit:

i. The Opcode field of the instruction has five bits available i.e. from bit 9 to 13. Hence this amounts to a total of 25

ii. As stated in the question that when operand 2 field is all zeros, each of original Opcodes takes on new meaning. This means 32 additional Opcodes if the value of operand 2 is all zero.

iii. Mode field consists of two bytes hence 22 possible. Theoretically speaking for each combination of mode field there is a whole class of Opcodes available.

iv. w/b field consists of one bit so only two possible values exist. Hence in this case also for each possible value of w/b field, there is a whole class of Opcodes available.

= 32 Opcodes.

= 4 different modes are Hence there comes a total of (32 + 32)*4*2 = 512 possible Opcodes. These Opcodes result from different combinations of mode field, w/b field and operand 2 field.

(b) Suggest an efficient way to provided more opcodes and indicate the trade-off involved.

Solution:
In order to provide room for additional Opcodes, we must sacrifice some functionality of the system. We can‟t reduce the field size of mode field as it will severely reduce the functionality of the system and the total Opcodes will also remain the same i.e. (64 + 64)

*2 * 2 = 512. Also the w/b field can‟t be reduced further as it is already 1 bit of size. We also can‟t reduce the size of operand fields because they have to access sixteen general purpose registers that is possible only through 4 bit value. Hence some other strategy must be considered.

As we have seen that operand 2 can‟t access general purpose register 0 this is because \if it has to access register 0 it will place its address as 0000 but this all zero code actually changes the meaning of original Opcodes. We can apply the same technique to operand 1 also. By making it not to use general purpose register 15 or R15 which is addressed as 1111 we can assign new meanings to all original Opcodes.

Hence by making the operand 1 field not use R15 and saying that if this field contains all 1s then all original Opcodes will have different meanings we can increase the number of Opcodes to (32 + 32 + 32)*4*2 = 768. So we have increased the number of Opcodes by 768 – 512 by 256 after sacrificing R15 from operand 1 field.

Q.12. A computer architect is designing a hardware datapath implementation and the architect has determined following circuit element delays.

Instruction Memory 150 ps
Decode 70 ps
Register Fetch 60 ps
ALU 150 ps
Data Memory 200 ps
Register Write Back 60 ps

(a) What is the length of a clock cycle for a single cycle datapath implementation?

Solution:
Time(lw) = Time(IF) + Time(ID+Reg.File) + Time(ALU) + Time(MemRead)
Time(lw) = 150 + (70+60) + 200 + 60
Time(lw) = 690ps

(b) What would be the frequency of a processor, corresponding to single datapath implementation?

Solution:
As we know Frequency =The value of clock cycle for single cycle is 1.
It means that frequency of processor corresponding to single cycle datapath is 1. Frequency can be defined as
Frequency = 1/clock rate+ Time(Reg.FileWrite)1GHz

c. where c is the maximum clock cycle.
=1/690* 10^-12*10^9 =1/690*10^ -3=1000/690= 1.44 9GHz=1.45GHz

(c) What would be the length of fastest clock cycle for a 5-stage pipeline datapath?
What would be the corresponding processor frequency?

Solution:
Fastest clock cycle is that whose latency is minimum i.e 60 ps. but when we calculate
the frequency, we have to consider the slowest cycle length in multi-cycle datapath that

is 200 ps.
So , Frequency = 1200 ps
1 -12 9 -3 *10 *10 =1/200*10 =1000/200=5 GHz

(d) How much faster is the 5-stage pipelined datapath compared to the single cycle datapath implementation?

Single cycle execution time= 690 ps
Execution time for multi cycle = 200ps
So 690/200ps=3.45 times faster

It means that 5-stage pipelined datapath is 3.45 time faster than single cycle datapath.
If we analyze this according to frequency point of view then 5GHz/1.45GHz=3.45 times faster

Q.13. What is the impact of increasing the size of branch-prediction buffer on two branches in a program?
Answer:

A single predictor predicting a single branch is generally more accurate than is that same predictor serving more than one instructions; and It is less likely that two branches in a program share a single predictor

Therefore, increasing the size of predictor buffer does not have significant effect on two branches in a program

Q.14. Consider the following mathematical expressions:
W= B << 3
X = 7B + B + C+ D
For each expression, write an assembly language code:

(a) For Reg-Reg architecture
(b) For Reg-Mem architecture

Solution:
W= B << 3
Reg-Reg Reg-Mem
LOAD R2, B
Shl [R2], 3
STORE R2, W
Shl [B], 3
STORE B, W
OR
Shl [R2] , 3
STORE R3,W
X = 7B + B + C+ D
Reg-Reg Reg-Mem
LOAD R2, B
LOAD R1, C
LOAD R3, D
ADD R4, R3, R1
ADD R5, R4, R2
MUL R2, R2, 7
ADD R4, R5,R2
STORE R4, X

Q.15. Consider a code:
i. Normalize the loop such that start index at 1 and increment it by 1 on every
ii. Write the normalized version of the loop then use GCD test to see if there is

For (i = 2 ; i < 100; i+ = 2)
A[i] = a[50 * i+1]iteration dependence.
Solution:
i. By normalize the loop, it leads to a modified c code sa shown below;
ii. The GCD test shows the potential for dependences written an array indexed

for (i = 1; i < 50; i++)
{

```
A[2 * i] = a [(100 * i) + 1] ;multiple constant by 2
}
```

by the function, Ai + b and ci + d only, If the condition (d-b) mod GCD(c, a) =0 is satisfied. Now, applying GCD test, in that case we will get, a=2, b=0, c=100 that allows us to determine dependence in loop. Thus, GCD will be, GCD(2, 100) = 2 and d-b=1. Here, as 1 is factor of 2. Thus, GCD test indicates that there is dependence in the code. In reality, there is no dependence in the code. Since the loop lead it value from a[101], a[201]....a[5001] and again these values to a[2], a[4].....a[100].

Q.16. Which approaches increase the amount of ILP?

Solution:

Loop unrolling, software pipelining, trace scheduling and superblocks approaches increase the amount of ILP, which can be exploited by a processor issuing more than one instruction on every clock cycle.

Q.17. Write down the reasons for which recurrence detection is important?

Solution:
Recurrence detection is important for two reasons:
1. Some architecture have special support for executing recurrences
2. Some recurrence can be the source of a reasonable amount of parallelism

Q.18. Find whether dependence exist in the following loop.

```
For (i = 1 ; i <100; i = i + 1)

{
X[2 * i + 3] = x[2 *i ] * 5.0;

}
```

Solution:
Here in X[2 * i + 3] such that a =2 and b =3 and in x[2 *i] such that c = 2 and d =0
Thus GCD(a, c) =2 and d-b=-3
Here, as 2 does not divide -3 so no dependence is possible

Q.**19.Define affine index?**

Solution:
An affine index is defined as follows;
An array index is affine if it can be written in the form of an expression. Here, a and b are constant, and i is the loop index variable. E.g. in the loop

```
For (i = 1 ; i <100; i = i + 1)

{
X[2 * i + 3] = x[2 *i ] * 5.0;
}
```

Here, the index value X[2 * i + 3]is affine with a = 2 and b = 3

Q.20.How compiler finds dependences and its assumption?

Solution:
The compiler detects the dependence using dependence analysis algorithm and this algorithm works on assumptions that:
- Array indices are affine
- There exist GCD of two affine indices

Q.21.Find the dependences in case of loop level parallelism(LLP)

Solution:

Here, two dependences occur. First, there exist dependence between the two uses of x[i] within the same iteration, so this is loop-level dependency but not loop carried.
Second, dependency occurs between the successive uses of i in different iterations which is loop-carried.
Moreover, there exist induction variable x[i]. therefore, dependence can be identified through compiler analysis near source level, and can be eliminated by loop unrolling.
for (i = 1000; i > 0; i = i – I)
X[i] = X[i] + s ;

Q.22.We have a program core consisting of five conditional branches. The program core will be executed thousands of times. Below are the outcomes of each branch for one execution of the program core (T for taken, N for not taken). (20).
Branch 1: T-T-T
Branch 2: N-N-N-N
Branch 3: T-N-T-N-T-N
Branch 4: T-T-T-N-T
Branch 5: T-T-N-T-T-N-T
Assume the behavior of each branch remains the same for each program core execution. For dynamic schemes, assume each branch has its own prediction buffer andeach buffer initialized to the same state before each execution. List the predictions for the following branch prediction schemes:
And also write down the prediction accuracies for each branch prediction scheme.

a) Always taken
b) Always not taken
c) 1-bit predictor, initialized to predict taken
d) 2-bit predictor, initialized to weakly predict taken

Solution:

Prediction accuracy = 100% * Correct Predictions / Total Branches

a) Branch 1: prediction: T-T-T, right = 3, wrong = 0
Branch 2: prediction: T-T-T-T, right = 0, wrong = 4
Branch 3: prediction: T-T-T-T-T-T, right = 3, wrong = 3
Branch 4: prediction: T-T-T-T-T, right = 4, wrong = 1
Branch 5: prediction: T-T-T-T-T-T-T, right = 5, wrong = 2
Total right = 15, Total wrong = 10, Accuracy = 100% * 15/25 = 60%

b) Branch 1: prediction: N-N-N, right = 0, wrong = 3
Branch 2: prediction: N-N-N-N, right = 4, wrong = 0
Branch 3: prediction: N-N-N-N-N-N, right = 3, wrong = 3
Branch 4: prediction: N-N-N-N-N, right = 1, wrong = 4
Branch 5: prediction: N-N-N-N-N-N-N, right = 2, wrong = 5
Total right = 10, Total wrong = 15, Accuracy = 100% * 10/25 = 40%

c) Branch 1: prediction: T-T-T, right = 3, wrong = 0
Branch 2: prediction: T-N-N-N, right = 3, wrong = 1
Branch 3: prediction: T-T-N-T-N-T, right = 1, wrong = 5
Branch 4: prediction: T-T-T-T-N, right = 3, wrong = 2
Branch 5: prediction: T-T-T-N-T-T-N, right = 3, wrong = 4
Total right = 13, Total wrong = 12, Accuracy = 100% * 13/25 = 52%

d) Branch 1: prediction: T-T-T, right = 3, wrong = 0
Branch 2: prediction: T-N-N-N, right = 3, wrong = 1
Branch 3: prediction: T-T-T-T-T-T, right = 3, wrong = 3
Branch 4: prediction: T-T-T-T-T, right = 4, wrong = 1
Branch 5: prediction: T-T-T-T-T-T-T, right = 5, wrong = 2
Total right = 18, Total wrong = 7, Accuracy = 100% * 18/25 = 72%

Q.22. What is dynamic Scheduling:
Dynamic Scheduling
• Hardware will detect and preserve dependencies (within a limited window of the instruction stream)
• Hardware will check for resource availability
• Independent instructions will be issued to the correct functional units TAG. Each tag identifies uniquely either ™ one of the 5 reservation stations ™ one of the 6 load buffers
• Indicates the “producer” of an operand that is not available from the registers
• A zero tag indicates that the operand is immediately available.

Q.24. What is the impact of increasing the size of branch-prediction buffer on two branches in a program?
Answer:

A single predictor predicting a single branch is generally more accurate than is that same predictor serving more than one instructions; and It is less likely that two branches in a program share a single predictor, Therefore, increasing the size of predictor buffer does not have significant effect on two branches in a program

Q.25 describe in detail the classification of I/O interconnects based on the communication distance,bandwidth latency and reliability.

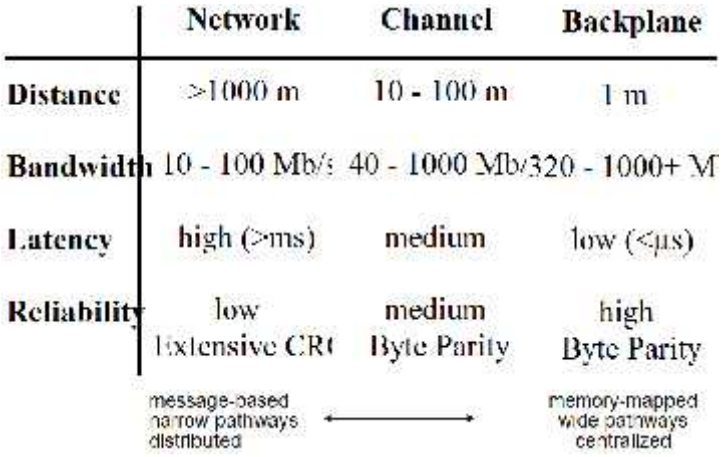
Ans; **Interconnect Trends**

• The I/O interconnect is the glue that interfaces computer system components

• I/O interconnects are facilitated using High speed hardware interfaces and logical protocols

• Based on the desired communication distance, bandwidth, latency and reliability, interconnects are classified as used:

• Backplanes, channels, Networks



Q.26: Suppose we have a processor with base CPI 1.0 , assuming all reference hit in the primary (level-1) cache and the clock rate 500 mhz. assume a main memory access time of 20 ns .including all miss handling . suppose the miss rate per instruction at the primary cache is 5%.

How much faster will the machine be if we add a second level cache that has a 20 ns access time for either a hit or miss and is large enough to reduce the miss rate to main memory to 2 %.

Answer.

Assume a computer has CPI=1.0 when all memory accesses are hit; the only data accesses are load/store access; and these are 50% of the total instructions

• If the miss rate is 2% and miss penalty is 25 clock cycles, how much faster the computer will be if all instructions are HIT

• Execution Time for all Hit = IC x 1.0 x cycle time
• CPU Execution time with real cache = CPU Execution time + Memory Stall time

• **Memory Stall Cycles =**
= IC x (Instruction access + data access) per instruction x miss rate x miss penalty
= IC (1+ 0.5) x 0.02 x 25
= IC x 0.75
• CPU Execution time (with cache) =
= (IC x 1.0 + IC x 0.75) x clock time
= 1.75 x IC x Cycle time Computer with no cache misses is 1.75 times faster

Q.27: Let us assume a computer has a 64 bytes cache block . an L2 cache that takes 7 clock cycle to get the critical 8 bytes, and then I clock cycle per 8 bytes +1 extra clock cycle to fetch the rast of the vlock without critical word first its 8 bytes clock for the first 8 bytes and the 1 clock per 8 bytes for the rest of the block.calculate the average penalty for critical word first assuming that there will be no other access to the rest of the block. Compare time with or without critical words first.

• An L2 cache takes 11 clock cycles to get first 8-byte (critical word) and then 2 clock cycles per 8- byte word to get the rest of the block (and 2 issues per cycle)

1. With critical word first (assuming no other access to the rest of the block)

2. Without critical word first (assuming following instructions read data sequentially 8-byte words at a time from the rest of the block; i.e., block load is required in this case)

Solution:

1. With Critical word first:
Average miss penalty
= Miss Penalty of critical word + Miss penalty of the remaining words of the block
= 11 x 1 + (8-1) x 2 = 11+ 14 = 25 clock cycles
2. Without critical word first (it requires block load)
= [Miss Penalty of first word + miss penalty of the remaining words of the block]
+ clock cycles to issue the load
= [11 x 1 + (8-1) x 2] + 8/4 = 25 + 4= 29 clock cycles
2 issues/cycle so 4cycles for 8 issues
• Merit: The merit of this technique is that it doesn‘t require extra hardware

• Drawback: This technique is generally useful only in large blocks, therefore the programs exhibiting spatial locality may face a problem is accessing the data or instruction from the memory, as the next miss is to the remainder of the block

Q.28. The running program pattern is 0x0 0x8 0x10 0x18 0x20 0x28

(a) If you directed mapped cache size 1KB and block size of 8 bytes (2 words)how many set:
(b) With the same cache and block size what is miss rate of the directed mapped.
(c) On which would decrease miss rate the most
  (i) Increasing the degree of associative by 2.
  (ii) Increasing the block size to 16 bytes.

Solution:

(a) Increasing block size will increase the cache’s ability to take advantage of spatial locality. This will reduce the miss rate for applications with spatial locality. However, it also decreases the number of locations to map an address, possibly increasing conflict misses. Also, the miss penalty (the amount of time it takes to fetch the cache block from memory) increases.

(b) Increasing the associativity increases the amount of necessary hardware but in most cases decreases the miss rate. Associativities above 8 usually show only incremental decreases in miss rate

Q.29. I/O Performance Parameters. 10 Marks Lecture # 38
I/O Performance Parameters

• Diversity: Which I/O device can connect to the CPU

• Capacity: How many I/O devices can connect to the CPU

• Latency: Overall response time to complete a task

• Bandwidth: Number of task completed in specified time - throughput

• The parameters diversity that refers to which I/O device and capacity means how many I/O devices can connect to the CPU are the I/O performance measures having no counterpart in CPU performance metrics.

• In addition, the latency (response time) and bandwidth (throughput) also apply to the I/O system.

• An I/O system is said to be in equilibrium state when the rate at which the I/O requests from CPU arriving, at the input of I/O queue (buffer) equals the rate at which the requests departs the queue after being fulfilled by the I/O device.

Q.30 Let us consider 32KB unified cache with misses per 1000 instruction equals 43.3 and instruction/data split caches each of 16KB with instruction cache misses per 1000 as 3.82 and data cache as 40.9. 15 Marks Lecture # 27 and 29 Cache Performance a). Find Average memory access time? b)

Solution: Miss Rate = (Misses/1000) / (Accesses/ inst.)
- Miss Rate 16KB Inst = (3.82/1000) /1.0 = 0.0038
- Miss Rate 16KB data = (40.9/1000) /0.36 = 0.114
- As about 74% of the memory access are instructions therefore overall miss rate for split caches = (74% x 0.0038) + (26% x 0.114) = 0.0324
- Miss Rate 32KB unified = (43.3/1000) /(1+0.36) = 0.0318
- i.e., the unified cache has slightly lower miss rate

2. Average Memory Access Time
= %inst x (Hit time + Inst. Miss rate x miss penalty)
+
%data x (Hit time + data Miss rate x miss penalty)
- Average Memory Access Time split

= 74% x (1 + 0.0038 x 100) + 26% x (1 + 0.114 x 100) = 4.24
- Average Memory Access Time unified

= 74%x (1 + 0.0.0318 x 100) + 26% x (1+1+0.0318 x 100) = 4.44
- i.e., the split caches have slightly better average access time and also avoids Structural Hazards

Q31. Let us consider a fully associative write-back cache with cache entries that start empty. Consider the following sequence of five memory operations and find, which address is not in the cache for no-write allocate. 10 Marks Lecture # 28

Write	Mem	[100]
Write	Mem	[100]
Read	Mem	[200]
Write	Mem	[200]

Write Mem [100]
- For no-write allocate, the address [100] is not in the cache (i.e., its tag is not in the cache
- So the first two writes will result in MISSES
- Address [200] is also not in the cache, the reed is also miss
- The subsequent write [200] is a hit
- The last write [100] is still a miss
- The result is 4 MISSes and 1 HIT
- For the write-allocate policy
- The first access to 100 and 200 are MISSES
- The rest are HITS as [100] and [200] are both found in the cache
- The result is 2 MISSes and 3 HITs
- Conclusion
  - Either write miss policy could be used with the write-through or write-back

Q4. If computer A runs a program 1 in 5 seconds and computer B runs the same program in 10 seconds and again Computer A Program 2 in 2 second and Computer B run same program in 1.5 second, which computer is faster and how much?

(5)

$$\text{Performance} \quad \frac{1}{\text{Execution Time}}$$

$$\text{Relative Performance} \quad \frac{\text{Performance}_x}{\text{Performance}_y} = \frac{\text{Execution Time}_y}{\text{Execution Time}_x} \quad \text{(aka speedup)}$$

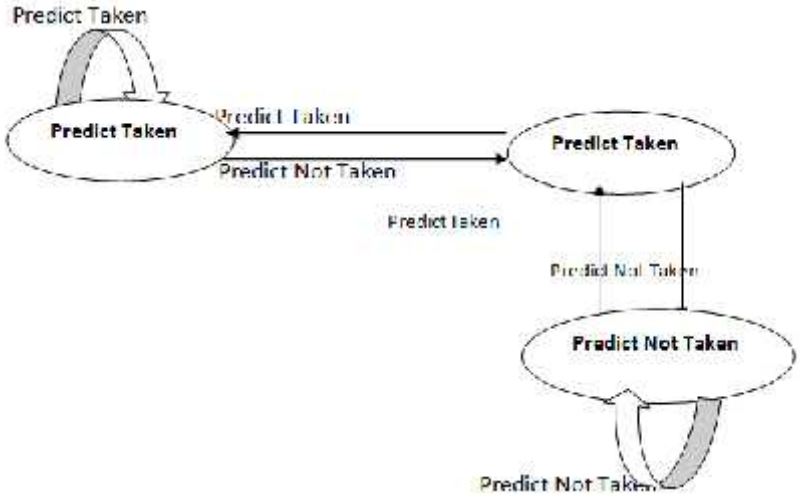
Example: time taken to run a program

- 10s on A, 15s on B
- Execution Time$_B$ / Execution Time$_A$ = 15s / 10s = 1.5
- So A is 1.5 times as fast as B (and B is 2/3 as fast as A)

Q4. Following state transition diagram represents a new branch prediction technique which is different from the standard 2-bit predictor. (Repeated=2)

(a) Describe the behavior of this new branch prediction method in detail. How it is different from the standard 2-bit predictor?

(b) Under what assumptions about the program behavior does this type of a branch prediction leads to better performance than a standard 2-bit predictor?



Solution:

(a) Describe the behavior of this new branch prediction method in detail. How it is different from the standard 2-bit predictor?

New branch prediction method's behavior is described as under:-

- Predictor mentioned in the diagram has two states; Predict Taken and Predict Not- Taken.
- Once the branch prediction is set to be as „predict taken" and later due to any reason it is „not taken", then the state will be remain as „predict taken". And if it is again „not taken" then the state will be changed to the state of predict „not-taken". This phenomenon of the predictor is same as the standard 2-bit predictor.
- Once the branch prediction is set to be as „not predict taken" and later due to any reason it is „predict taken", then the state changes to as „predict taken". This phenomenon of the predictor is different from the standard 2-bit predictor. In standard 2- bit predictor, it will be wrong if the state is set at „not-taken" and at least for two times before its state is changed to the „predict taken".

- Therefore in this predictor, the decision has to be wrong for two times if it takes„predict taken" to „predict not taken" before prediction is reversed. But when the decision has to be wrong for one time if it takes „predict not taken" to „predict taken"‟ before prediction is reversed.
- Hence this predictor appears to be in favor of the prediction that „branch will be taken".

(b) Under what assumptions about the program behavior does this type of a branch prediction leads to better performance than a standard 2-bit predictor?

Solution:

In the following circumstances, the branch prediction does lead to better performance than a standard 2-bit predictor:-

1. Looped Programs

a. Any program which have a single or number of loops must take more time than normal.

b. Loops with variable lengths repeat the instruction unless required result is not achieved.

c. Here due to the repetition of instruction in a loop there will be one wrong guess of the state as compared to the 2-bit predictor where two wrong predictions are guessed before changing its state.

d. If a loop has 500 repetitions then the predictor will guess out 499 times accurately (99.8%.)

2. Conditional Programs

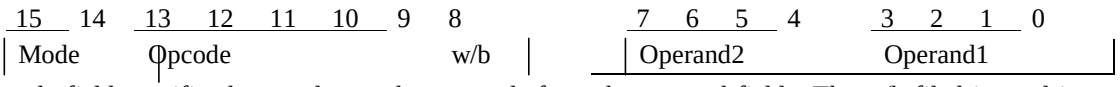
a. Those programs which have some conditions like „if" „if else"‟ etc have more chances to get benefit from this new design of predictor.

b. The predictor if sets at the „taken" state then there are chances that the performance will be enhanced due the predictor"s property of one wrong guess.

----------SOLUTIONS----------

Question 1:

The 16-bit Zilog Z8001 has the following general instruction format:



The mode field specifies how to locate the operands from the operand fields. The w/b filed is used in certain instructions to specify whether the operands are byte or 16-bit words. The operand 1 field may (depending on the mode field contents) specify one of

16 general-purpose registers. The operand 2 field may specify any general-purpose registers except register 0. When the operand 2 field is all zeros, each of the original opcodes takes on a new meaning.

(a) How many opcodes are provided on the Z8001?

Solution:

Let us analyze the problem a bit:

i. The Opcode field of the instruction has five bits available i.e. from bit 9 to 13. Hence this amounts to a total of 2^5 = 32 Opcodes.

ii. As stated in the question that when operand 2 field is all zeros, each of original Opcodes takes on new meaning. This means 32 additional Opcodes if the value of operand 2 is all zero.

iii. Mode field consists of two bytes hence 2^2 = 4 different modes are possible. Theoretically speaking for each combination of mode field there is a whole class of Opcodes available.

iv. w/b field consists of one bit so only two possible values exist. Hence in this case also for each possible value of w/b field, there is a whole class of Opcodes available.

Hence there comes a total of (32 + 32)*4*2 = 512 possible Opcodes. These Opcodes result from different combinations of mode field, w/b field and operand 2 field.

(b) Suggest an efficient way to provided more opcodes and indicate the trade-off involved.

Solution:

In order to provide room for additional Opcodes, we must sacrifice some functionality of the system. We can‟t reduce the field size of mode field as it will severely reduce the functionality of the system and the total Opcodes will also remain the same i.e. (64 + 64)
*2 * 2 = 512. Also the w/b field can‟t be reduced further as it is already 1 bit of size. We also can‟t reduce the size of operand fields because they have to access sixteen general purpose registers that is possible only through 4 bit value. Hence some other strategy must be considered.
As we have seen that operand 2 can‟t access general purpose register 0 this is because if it has to access register 0 it will place its address as 0000 but this all zero code actually changes the meaning of original Opcodes. We can apply the same technique to operand 1 also. By making it not to use general purpose register 15 or R15 which is addressed as 1111 we can assign new meanings to all original Opcodes
Hence by making the operand 1 field not use R15 and saying that if this field contains all
1s then all original Opcodes will have different meanings we can increase the number of
Opcodes to (32 + 32 + 32)*4*2 = 768. So we have increased the number of Opcodes by
768 – 512 by 256 after sacrificing R15 from operand 1 field.

Question 2:

A computer architect is designing a hardware datapath implementation and the architect has determined following circuit element delays.

Instruction Memory	150 ps
Decode	70 ps
Register Fetch	60 ps
ALU	150 ps
Data Memory	200 ps
Register Write Back	60 ps

(a) What is the length of a clock cycle for a single cycle datapath implementation?
Solution:

Time(lw) = Time(IF) + Time(ID+Reg.File) + Time(ALU) + Time(MemRead) + Time(Reg.FileWrite)

Time(lw) = 150 + (70+60) + 200 + 60
Time(lw) = 690ps

(b) What would be the frequency of a processor, corresponding to single datapath implementation?
Solution:
As we know

Frequency = $\frac{1}{c}$ *GHz* where c is the maximum clock cycle.

The value of clock cycle for single cycle is 1. It means that frequency of processor corresponding to single cycle datapath is 1. Frequency can be defined as

Frequency = 1/clock rate

=1/690* 10^$^{-12}$*10^^{9} =1/690*10^ -3=1000/690= 1.44 9GHz=1.45GHz

(c) What would be the length of fastest clock cycle for a 5-stage pipeline datapath? What would be the corresponding processor frequency?

Solution:

Fastest clock cycle is that whose latency is minimum i.e 60 ps. but when we calculate the frequency, we have to consider the slowest cycle length in multi-cycle datapath that is 200 ps.

Frequency = So $\frac{1}{200\ ps}$

Frequency = $\frac{1}{200}$*10^{-12} *10^{9} =1/200*10^{-3} =1000/200=5 GHz

(d) How much faster is the 5-stage pipelined datapath compared to the single cycle datapath implementation?
Single cycle execution time= 690 ps Execution time for multi
cycle = 200ps So 690/200ps=3.45 times faster
It means that 5-stage pipelined datapath is 3.45 time faster than single cycle datapath.
If we analyze this according to frequency point of view then
5GHz/1.45GHz=3.45 times faster

Question 3:

Consider the following mathematical expressions: W= B << 3

X = 7B + B + C+ D

For each expression, write an assembly language code:

(a) For Reg-Reg architecture

(b) For Reg-Mem architecture

W= B << 3	
Reg-Reg	Reg-Mem
LOAD R2, B Shl [R2], 3 STORE R2, W	Shl [B], 3 STORE B, W OR Shl [R2] , 3 STORE R3,W
X = 7B + B + C+ D	
Reg-Reg	Reg-Mem

LOAD R2, B
LOAD R1, C
LOAD R3, D ADD
R4, R3, R1
ADD R5, R4, R2
MUL R2, R2, 7
ADD R4, R5,R2
STORE R4, X
	LOAD R2, B
	ADD R1, R2, C
	ADD R3, R1, D
	MUL R4, R2, 7
	ADD R4, R4, R3
	STORE R4 , X

Question 4:

Following code lines are written in a high level language:

a = c + d;

b = c + e;

The corresponding instructions for MIPS are:

These instructions are to be executed on a pipelined processor with forwarding.

(a) Identify hazards by showing the execution of these instructions per cycle basis.

No	Inst. / Cycle	1	2	3	4	5	6	7	8	9	10	11
1	**LW R1, 0(R0)**	Im	Reg	ALU	Dm	Reg						
2	**LW R2, 4(R0)**		Im	Reg	ALU	Dm	Reg					
3	**ADD R3, R1, R2**			Im	Reg	ALU	Dm	Reg				
4	**SW R3, 12(R0)**				Im	Reg	ALU	Dm	Reg			
5	**LW R4, 8(R0)**					Im	Reg	ALU	Dm	Reg		
6	**ADD R5, R1, R4**						Im	Reg	ALU	Dm	Reg	
7	**SW R5, 16(R0)**							Im	Reg	ALU	Dm	Reg

Both add instructions have a hazard because of their respective dependence on the immediately preceding

(b) Reorder these instructions to avoid any pipeline stalls.

No	Inst. / Cycle	1	2	3	4	5	6	7	8	9	10	11
1	**LW R1, 0(R0)**	Im	Reg	ALU	Dm	Reg						
2	**LW R2, 4(R0)**		Im	Reg	ALU	Dm	Reg					
3	**SW R3, 12(R0)**			Im	Reg	ALU	Dm	Reg				
4	**ADD R3, R1, R2**				Im	Reg	ALU	Dm	Reg			
5	**LW R4, 8(R0)**					Im	Reg	ALU	Dm	Reg		
6	**ADD R5, R1, R4**						Im	Reg	ALU	Dm	Reg	
7	**SW R5, 16(R0)**							Im	Reg	ALU	Dm	Reg

(c) How many cycles are saved after executing the reordered instructions?

On a pipelined processor with forwarding, the reordered sequence will complete in two fewer cycles than the original version.

Forwarding yields another insight into the MIPS architecture. Each MIPS instruction writes at most one result and does this in the last stage of the pipeline. Forwarding is harder if there are multiple results to forward per instruction or they need to write a result early on in instruction execution.

Question 5:

Following code segment is written in a high level language:

```
int randArray[1000] = { /* initialized with some random integers */ };
int sum1 = 0, sum2 = 0, sum3 = 0, sum4 = 0;
for (int i = 0; i < 1000; i ++)     // Branch_1: Loop Branch {
        if (i % 4 == 0)                         // Branch_2: If Condition 1 sum1 =
                sum1 + randArray[i];  // Taken Path
         else
                sum2 = sum2 + randArray[i];  // Not-Taken Path
        if (i % 2 == 0)                         // Branch_3: If Condition 2 sum3 =
                sum3 + randArray[i];  // Taken Path
         else
                sum4 = sum4 + randArray[i];  // Not-Taken Path
                                        }
```

(a) Find the prediction accuracy for all the branches using a separate last-time branch predictor for every branch (initial value of each branch counter is "not-taken"). Show all steps.

Answer (a):
Under the last-time branch predictor, the prediction accuracy for each branch is shown in the followings:

Prediction accuracy for Branch_1: Loop Branch
This branch is a loop. The loop will run 1001 times. Initial value of the branch predictor is „not taken" but the branch will be taken. So this is one wrong prediction and the predictor will set its value from „not taken" to taken. The prediction will be correct when the value goes from 1 to 999. But when the value reaches 1000 then prediction is again wrong. So the predictor is only 2 times wrong out of 1001 times. So the accuracy = (999/1001)*100 = 99.8 % for Branch_1

Prediction accuracy for Branch_2: If condition 1
This branch is an if-else branch. As this branch is nested inside the loop so it will run 1000 times. Initially the predictor for this branch is set to „not-taken".

- 0 % 4 = 0
  Predictor is set to „not-taken". Branch is taken. Wrong prediction occurred. Predictor is set to „taken"
- 1 % 4 = 1
  Predictor is set to „taken". Branch is not taken. Wrong prediction occurred. Predictor is set to „not-taken"
- 2 % 4 = 2
  Predictor is set to „not-taken". Branch is not taken. Correct prediction occurred. Predictor does not change
- 3 % 4 = 3
  Predictor is set to „not-taken". Branch is not taken. Correct prediction occurred. Predictor does not change

Here the predictor is set to „not-taken" and the next modulus is again going from 0, 1, 2, 3, 0, 1, 2, 3 … So the above behavior will be repeated i.e. 2 out of 4 prediction will be correct.
So the accuracy = (500/1000)*100 = 50 % for Branch_2

Prediction accuracy for Branch_3: If condition 2
This branch is also an if-else branch. As this branch is also nested inside the loop so it will run 1000 times. Initially the predictor for this branch is set to „not-taken".

- 0 % 2 = 0
  Predictor is set to „not-taken". Branch is taken. Wrong prediction occurred. Predictor is set to „taken"
- 1 % 2 = 1
  Predictor is set to „taken". Branch is not taken. Wrong prediction occurred. Predictor is set to „not-taken"
- 2 % 2 = 0

  Predictor is set to „not-taken". Branch is taken. Wrong prediction occurred. Predictor is set to „taken"
- 3 % 2 = 1
  Predictor is set to „taken". Branch is not taken. Wrong prediction occurred. Predictor is set to „not-taken,

The same process goes on and the predictor every times makes a wrong prediction. So the accuracy = (0/1000)*100 = 0 % for Branch_3

(b) Find the prediction accuracy for all the three branches using a 2-bit saturating counter-based predictor for every branch (initial value of each branch counter is "strongly not-taken"). Show all steps.

Answer:
Under the 2-bit saturating counter-based predictor, the prediction accuracy for each branch is shown in the followings

Prediction accuracy for Branch_1: Loop Branch
This branch is a loop. The loop will run 1001 times. Initial value of the branch predictor is

„strongly not taken".

- At first iteration the branch will be taken. Wrong prediction. The state will be changed from „strongly not taken" to „weakly not taken".
- At 2nd iteration the branch will be taken. Wrong prediction. The state will be changed from „weakly not taken" to „weakly taken".
- All the next prediction up to i<1000 will be correct
- At 1001th iteration (when i=1000) the branch will not be taken. Wrong prediction. So the

accuracy = (998/1001)*100 = 99.7% for Branch_1

Prediction accuracy for Branch_2: If condition 1

This branch is an if-else branch. As this branch is nested inside the loop so it will run 1000 times. Initially the predictor for this branch is set to „strongly not-taken".

- 0 % 4 = 0
  Predictor is set to „strongly not-taken". Branch is taken. Wrong prediction occurred. Predictor is set to „weakly not taken"
- 1 % 4 = 1
  Predictor is set to „weakly not taken". Branch is not taken. Correct prediction occurred. Predictor is set to „strongly not-taken"
- 2 % 4 = 2
  Predictor is set to „strongly not-taken". Branch is not taken. Correct prediction occurred. Predictor does not change
- 3 % 4 = 3
  Predictor is set to „strongly not-taken". Branch is not taken. Correct prediction occurred. Predictor does not change

Here the predictor is set to „strongly not-taken" and the next modulus is again going from 0, 1, 2, 3, 0, 1, 2, 3 … So the above behavior will be repeated i.e. 1 out of 4 prediction will be correct.

So the accuracy = (750/1000)*100 = 75 % for Branch_2

Prediction accuracy for Branch_3: If condition 2

This branch is also an if-else branch. As this branch is also nested inside the loop so it will run 1000 times. Initially the predictor for this branch is set to „strongly not-taken".

- 0 % 2 = 0
  Predictor is set to „strongly not-taken". Branch is taken. Wrong prediction occurred. Predictor is set to „weakly not taken"
- 1 % 2 = 1
  Predictor is set to „weakly not taken". Branch is not taken. Correct prediction occurred. Predictor is set to „strongly not-taken"
- 2 % 2 = 0
  Predictor is set to „strongly not-taken". Branch is taken. Wrong prediction occurred. Predictor is set to „weakly not taken"
- 3 % 2 = 1
  Predictor is set to „weakly not taken". Branch is not taken. Correct prediction occurred. Predictor is set to „strongly not-taken,

The same process goes on and the predictor every times makes 1 correct prediction out of 2.
So the accuracy = (500/1000)*100 = 50 % for Branch_3

Question 6:

Compare zero-, one-, two- and three-address machines by writing programs to compute: X = (A + B x C) / (D - E x F) for each of the four machines. The instructions available for use are:

0-Address	1-Address	2-Address	3-Address
PUSH M POP M ADD SUB MUL DIV	LOAD M STORE M ADD M SUB M MUL M DIV M	MOVE (X □ Y) ADD (X □ X + Y) SUB (X □ X - Y) MUL (X □ X x Y) DIV (X □ X / Y)	MOVE (X □ Y) ADD (X □ Y + Z) SUB (X □ Y - Z) MUL (X □ Y x Z) DIV (X □ Y / Z)

Solution:

0-Address Instructions: These instructions used stack to hold both operands and result. Stack based architecture is not required any address because two operands are always present at the top of stack. Operations are performed between values on the top of the stack (TOS) and second value on the stack (SOS). The result always stored on the TOS.

1-Address Instructions: These instructions used accumulator based architecture. Accumulator is used to hold the operand and result .LOAD instruction loads the value of variable in accumulator and STORE instruction stores the value of accumulator in memory.

. **2-Address Instructions**: 2-Address instructions used general purpose register to holds one of the operands and result. In order to move the value of variable in register or value of register to the memory we used MOVE instruction .R1 and R2 register is used.

3-Address Instructions: Maximum three operands are allowed in these instructions that may be memory locations or registers.

The given expression X = (A + B x C) / (D - E x F), can be solved by the following sequence:

DIV ;pop out the result from TOS andstore it as memory X POP X	ADD H ;divides thevalue ;of acc with S DIV S ; stores value ofacc as X i.e. result STORE X		

Question 7:

Consider the results of question_1. Assume that M is a 16-bit memory address and that X, Y, and Z are either 16-bit addresses or 4-bit register numbers. The one-address machine uses an accumulator, and the two- and three-address machines have 16 registers and instructions operating on all combinations of memory locations and registers. Assuming 8-bit opcodes and instruction lengths that are multiples of 4 bits, how many bits does each machine need to compute X?

Solution:

1. **0-Address Instructions**: As given in question that opcode is 8 bit long and 16 bit memory address . Hence the maximum length of instruction is 24 bits. There are a total of 12 instructions hence 12 * 24 = 228 bits are required by the machine to compute it.
2. **1-Address Instructions** As given in question that opcode is 8 bit long and 16 bit memory address . Hence the maximum length of instruction is 24 bits. There are a total of 13 instructions. Hence total 13 * 24 = 312 bits are required by the machine to compute it.
3. **2-Address Instructions**: We compute the bits as follows:

Instruction	**Bits**
MOVE R1, E	8 + 4 + 8 = 20
MUL R1, F	8 + 4 + 8 = 20
MOVE R2, D	8 + 4 + 8 = 20
SUB R2, R1	8 + 4 + 4 = 16
MOVE R2, Y	8 + 4 + 8 = 20
MOVE R1, B	8 + 4 + 8 = 20
MUL R1, C	8 + 4 + 8 = 20
MOVE R2, A	8 + 4 + 8 = 20
ADD R1, R2	8 + 4 + 4 = 16
MOVE R2, Y	8 + 4 + 8 = 20
DIV R1, R2	8 + 4 + 4 = 16
MOVE R1, X	8 + 4 + 8 = 20
Total	**228 bits**

4. **3-Address Instructions**:

Instruction	**Bits**
MUL R1, E, F	8 + 4 + 8 + 8 = 28
SUB R1, D, R1	8 + 4 + 8 + 4 = 24
MUL R2, B, C	8 + 4 + 8 + 8 = 28
ADD R2, A, R2	8 + 4 + 8 + 4 = 24
DIV R1, R2, R1	8 + 4 + 4 + 4 = 20
MOVE X, R1	8 + 8 + 4 = 20
Total	**144 bits**

Question 8: Consider the following mathematical expressions:

U = A + B + D V
= C + D
W= B << 3
X = 7B + B + C+ D Y
= X + V

For each expression, write an assembly language code: (a)
For Reg-Reg architecture
(b) For Reg-Mem architecture

REG-REG Architecture	**REG-MEM Architecture**
U = A + B + D	
LOAD R1, A LOAD R2, B LOAD R3 ,D ADD R4, R3,R2 ADD R4, R4,R1 STORE R4,U	MOV R1, A ADD R4, R1, B ADD R5, R4, D STORE R5, U
V = C + D	
LOAD R4, C LOAD R3, D ADD R4, R3,R4 STORE R4, V	LOAD R4, C ADD R4, R4, D STORE R4, V
W= B << 3	
LOAD R2, B Shl [R2], 3 STORE R2, W	Shl [B], 3 STORE B, W OR Shl [R2] , 3 STORE R3,W
X = 7B + B + C+ D	

LOAD R2, B
LOAD R1, C
LOAD R3, D ADD R4, R3, R1
ADD R5, R4, R2
MUL R2, R2, 7
	LOAD R2, B
	ADD R1, R2, C
	ADD R3, R1, D
	MUL R4, R2, 7
	ADD R4, R4, R3
	STORE R4 , X

ADD R4, R5,R2 STORE R4, X	
Y = X + V	
LOAD R1, X LOAD R3, V ADD R2, R1, R3 STORE R2, Y	LOAD R1, X ADD R2, R1, V STORE R2, Y

Question 9:

Consider the following code:

```
LD R1, 0(R2)        ; load R1 from address 0+R2
DADD R4, R1, R5     ; R4 =R1 + R5
SD R5, 100(R1)      ; store R5 at address 100+R1
DSUB R3, R1, R4     ; R3 = R1   - R4
```

(a) Identify all the data dependencies in this code by showing the execution of these instructions per cycle bases. Which of these dependencies are data hazards and will be resolved via forwarding? (Conventional 5-stage RISC pipeline)

Solution:

In the given code two types of dependency exists

Data Dependency

There four data dependency in the given code.

1. As first instruction loads R1 from address 0+R2, the value of R1 is needed for 2nd instruction that is *DADD R4, R1, R5*. It means there is dependency between these two instructions.
2. Second dependency is between SD R5, 100(R1) and LD R1, 0(R2) because the value of R1 is required for SD instruction which is not available at time. So there is data dependency between these two instruction.
3. Third dependency also exists between DADD R4, R1, R5 and DSUB R3, R1, R4 on R4 because the value of R4 is required for DSUB but it is not available on time so we have to use stall.
4. There is data dependency between LD and DSUB instruction on R1. Since DSUB required value earlier but it is not available.

Name Dependency

In the given code only one name dependency exist which does not affect the result that is on R5 between DADD R4, R1, R5 and SD R5, 100(R1).

Timing diagram

Instructions	Clock cycles												
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**		
LD R1, 0(R2)	IF	ID	Ex	MEM	WB								
DADD R4, R1, R5		IF	ID	stall	Ex	MEM	WB						
SD R5, 100(R1)			IF	stall	ID	Ex	MEM	WB					
DSUB R3, R1, R4				Stall	IF	ID	Ex	MEM	WB				

(b)The timing diagram shown in part (a) is in fact the one when no forwarding or bypassing is used and each memory reference takes one clock cycle. So to execute the code 11 clock cycles are required in the said case.

(c) If the normal data forwarding and bypassing is used then the timing diagram would be as under:

	Clock Cycles											
Instructions	1	2	3	4	5	6	7	8	9	10	11	12
LD R1, 0(R2)	IF	ID	EX	MEM	WB							
DADD R4, R1, R5		IF	ID	Stall	EX	MEM	WB					
SD R5, 100(R1)			IF	Stall	ID	EX	MEM	WB				
DSUB R3, R1, R4					IF	ID	EX	MEM	WB			

So, it will take 9 clock cycles to execute the code if normal data forwarding and bypassing is used.

Question 10:

Following code segment is written in a high level language:

```
int randArray[1000] = { /* initialized with some random integers */ };
int sum1 = 0, sum2 = 0, sum3 = 0, sum4 = 0;
for (int i = 0; i < 1000; i ++)     // Branch_1: Loop Branch
{
        if (i % 4 == 0)  // Branch_2: If
        Condition 1 sum1 = sum1 +
        randArray[i];  // Taken Path else
        sum2 = sum2 + randArray[i];  // Not-
        Taken Path if (i % 2 == 0)  // Branch_3:
        If Condition 2
        sum3 = sum3 + randArray[i];  // Taken Path
        else
        sum4 = sum4 + randArray[i];  // Not-Taken Path
}
```

(a) Find the prediction accuracy for all the branches using a separate last-time branch predictor for every branch (initial value of each branch counter is "not-taken"). Show all steps.

Solution:

In order to know the prediction accuracy for all branches let us discuss all branches one by one.

Branch 1: This branch is basically a loop branch. In this loop it is predict that will be taken or not taken. Of course loop will execute according to limit given by loop variable. The loop will be taken till the value of loop is within given limit. In the above code loop will execute 1000 times. As mentioned in the question that the initial value is not taken so this will be first error because the predictor will predict that loop will not taken but infect it executes. So the value is reversed from not taken to taken.

We know that predictor is predicting loop taken now but after limit cross the predictor is again predict the loop taken but infect it is not taken because increment variable meet the limit so it will be second wrong prediction of predictor and value reversed from taken to not taken.

If we observe above prediction then we can see that there are only two wrong prediction made by predictor. It means that 998 time loop prediction was accurate but only two times were wrong.

Hence accuracy for branch 1 is 998/1000=99.8 %

Branch 2: This is basically an "if" statement which will execute on the base of value of i of loop. "if" statement will execute on the base of value of control variable i of loop by taking its modulus 4 that is i%4 , if its value is equal to equal to zero then the path taken. The modulus will be zero when it is multiple of 4 that is 0,4,8,12,…….

On the base of above detail, I conclude the following result

Iteration(i)	Branch	Prediction	Bit	Fault	Error
i=0	taken	incorrect	Reversed	occur	Yes
i=1	Not taken	incorrect	reversed	Occur	Yes
i=2	Not taken	Correct	Not reversed	Not occur	No
i=3	Not taken	Correct	Not reversed	Not occur	No
i=4	Taken	Incorrect	Reversed	occur	Yes
i=5	Not taken	Incorrect	Reversed	occur	Yes
i=6	Not taken	Correct	Not reversed	Not occur	No
i=7	Not taken	correct	Not reversed	Not occur	no

We observe from above table that for 8 iterations , four times prediction is correct and 4 times not correct. It means that accuracy is 50%. So for 1000 iterations prediction 500 times will be correct and 500 times will be incorrect.

Branch 3:

This is basically an "if" statement which will execute on the base of value of i of loop. "if" statement will execute on the base of value of control variable i of loop by taking its modulus 2 that is i%2 , if its value is equal to equal to zero then the path taken. The modulus will be zero when it is multiple of 4 that is 0,2,4,6,8, 10, 12, … .,…….

On the base of above detail, I conclude the following result

Iteration(i)	Branch	Prediction	Bit	Fault	Error
i=0	taken	incorrect	Reversed	occur	Yes
i=1	Not taken	incorrect	reversed	occur	Yes
i=2	taken	incorrect	reversed	occur	Yes
i=3	Not taken	incorrect	reversed	occur	Yes
i=4	Taken	Incorrect	reversed	occur	Yes

We observe from above table that for 4 iterations , prediction(guess) is always wrong. It means that accuracy is 0%. So for 1000 iterations prediction 1000 times will be correct wrong.

(b) Find the prediction accuracy for all the three branches using a 2-bit saturating counter- based predictor for every branch (initial value of each branch counter is "strongly not- taken"). Show all steps.

Solution:

In order to know the prediction accuracy for all branches let us discuss all branches one by one.

Branch 1: This branch is basically a loop branch. In this loop it is predict that will be taken or not taken. Of course loop will execute according to limit given by loop variable. The loop will be taken till the value of loop is within given limit. In the above code loop will execute 1000 times. As mentioned in the question that the initial value is not taken so this will be first error because the predictor will predict that loop will not taken but infect it executes. So the state of predictor changes to weakly not taken.

The second error occurs at iteration two when predictor predict that loop will not run but infect it execute. The state of predictor changes to weakly taken.

We know that predictor is predicting loop taken now but after limit cross the predictor is

again predict the loop taken but infect it is not taken because increment variable meet the limit so it will be third wrong prediction of predictor and state of predictor changes to weakly taken. If we observe above prediction then we can see that there are only three wrong prediction made by predictor. It means that 997 time loop prediction was accurate but only two times were wrong.

Hence accuracy for branch 1 is 997/1000=99.7 %.

Branch2:

This is basically an “if” statement which will execute on the base of value of i of loop. “if” statement will execute on the base of value of control variable i of loop by taking its modulus 4 that is i%4 , if its value is equal to equal to zero then the path taken. The modulus will be zero when it is multiple of 4 that is 0,4,8,12,…….

On the base of above detail, I conclude the following result

Iteration(i)	Branch	Prediction	State	Fault	Error
i=0	taken	correct	Weekly not taken	occur	Yes
i=1	Not taken	incorrect	Strongly not taken	Not occur	No
i=2	Not taken	Correct	Strongly	Not occur	No
			not taken		
i=3	Not taken	Correct	Strongly not taken	Not occur	No
i=4	Taken	Incorrect	Weekly not taken	occur	Yes
i=5	Not taken	correct	Strongly not taken	Not occur	No
i=6	Not taken	Correct	Strongly not taken	Not occur	No

i=7	Not taken	correct	Strongly not taken	Not occur	no

We observe from above table that for 8 iterations , 6 times prediction is correct and 2 times not correct. It means that accuracy is 75%. So for 1000 iterations prediction 750 times will be correct and 250 times will be incorrect. So 750/1000=75%.

Branch 3:

This is basically an "if" statement which will execute on the base of value of i of loop. "if" statement will execute on the base of value of control variable i of loop by taking its modulus 2 that is i%2 , if its value is equal to equal to zero then the path taken. The modulus will be zero when it is multiple of 4 that is 0,2,4,6,8, 10, 12, … .,…….

On the base of above detail, I conclude the following result

Iteration(i)	Branch	Prediction	state	Fault	Error
i=0	taken	incorrect	Weekly not taken	occur	Yes
i=1	Not taken	correct	Strongly not taken	Not occur	Yes
i=2	taken	incorrect	Weekly not taken	occur	Yes
i=3	Not taken	correct	Strongly not taken	Not occur	Yes
i=4	Taken	Incorrect	Weekly not taken	occur	Yes

We observe from above table that for 4 iterations , two times prediction is correct and two times not correct. It means that accuracy is 50%. So for 1000 iterations prediction
500 times will be correct and 500 times will be incorrect.

Question 11:

MIPS chooses to simplify the structure of its instructions. The way we implement complex instructions through the use of MIPS instructions is to decompose such complex instructions into multiple simpler MIPS ones. Show how MIPS can implement the instruction *swap $rs, $rt*, which swaps the contents of registers *$rs* and *$rt*. Consider the case in which there is an available register that may be destroyed as well as the care in which no such register exists.

If the implementation of this instruction in hardware will increase the clock period of a single-instruction implementation by 10%, what percentage of swap operations in the
instruction mix would recommend implementing it in hardware?

Solution:

a. **Swapping when a register is available**: Suppose we have a register $rz that is available for swapping operation and which will be destroyed at the end of operation then the sequence of instructions will be as follows:

```
MOVE $rz, $rs     ; moves the contents of $rs to $rz
MOVE $rs, $rt     ; moves the contents of $rt to $rs
MOVE $rt, $rz     ; moves the contents of $rz in $rt
```

b. **Swapping when no register is available**: When there is no register available for swapping, there are two possibilities in which we can accomplish this task i.e. through register to memory transfer operations or through XOR algorithm. We touch both of them:

i. **Register to Memory**:

```
SW $rs, X     ; stores word from $rs to X location in memory SW $rt, Y
; stores word from $rt to Y location in memory LW $rs, Y     ; loads
word from Y location to $rs
LW $rt, X     ; loads word from X location to $rt ii.
```

XOR algorithm:

```
XOR $rs, $rs, $rt
XOR $rt, $rs, $rt
XOR $rs, $rs, $rt
```

c. Since implementation of instruction at hardware level calls for 10% increases in clock period. That means that if 10% of instructions in the instruction mix are swap instruction then there will be 1% increase in clock period and if there are 20% instructions then there will be 2% increase. So in our opinion there must not be more than 30% swap instructions in the instructions mix.

Question 12:

Compare zero-, one-, two- and three-address machines by writing programs to compute: X = (A + B x C) / (D - E x F)
for each of the four machines. The instructions available for use are:

0-Address	1-Address	2-Address	3-Address
PUSH M	LOAD M	MOVE (X □ Y)	MOVE (X □ Y)
POP M	STORE M	ADD (X □ X + Y)	ADD (X □ Y + Z)
ADD SUB	ADD M	SUB (X □ X - Y)	SUB (X □ Y - Z)
MUL	SUB M	MUL (X □ X x Y)	MUL (X □ Y x Z)
DIV	MUL M	DIV (X □ X / Y)	DIV (X □ Y / Z)
	DIV M		

Solution:

a. **0-Address Instructions**: 0-Address instructions are present in stack based architecture. The two operands are always present at the top of the stack so no address is to be given. Result is also placed at top of stack: in order to solve the expression X = (A + B x C) / (D - E x F), we give the sequence of instructions as:

```
PUSH A      ; pushes A on stack
PUSH B      ; pushes B on stack
PUSH C      ; pushes C on stack
MUL         ; multiplies the two operands on top of stack i.e. B, C,
            ; result stored on top of stack (TOS).
ADD         ; adds the two operands on top of stack i.e. result of
            ; previous instruction and A result stored on TOS
PUSH D      ; pushes D on stack PUSH
E           ; pushes E on stack PUSH F
; pushes F on stack
MUL         ; multiplies the two operands on TOS i.e. E, F result TOS SUB
; subtracts the two operands on TOS i.e. result of last
            ; instruction and D result stored on TOS
DIV         ; divides the two operands on TOS, result stored on TOS POP X
; pop out the result from TOS and store it as memory X
```

b. **1-Address Instructions**: 1-Address instructions are present in accumulator based architecture. In this kind, one of the operand is always present in accumulator and the result is also stored in accumulator. LOAD instruction loads the value of variable in accumulator and STORE instruction stores the value of accumulator in memory. In order to solve the expression X = (A + B x C) / (D - E x F), we give the sequence of instructions as:

```
LOAD F      ; loads accumulator (acc) with F
MUL E       ; multiplies the value of acc with E, result
            ; stored in acc
STORE G     ; stores the value of acc as a memory location G LOAD D
; loads acc with D
SUB G       ; subtracts the value of G from value of acc STORE S    ;
stores the value of acc as memory location S LOAD C      ; loads acc
with C
MUL B       ; multiplies the value of acc with B
STORE H     ; stores the value of acc as memory location H LOAD A
; loads the acc with value of A
ADD H       ; adds the value of H in value of Acc
DIV S       ; divides the value of acc with S
STORE X     ; stores the value of accumulator as X i.e. result
```

c. **2-Address Instructions**: 2-Address instructions involve a general purpose register which holds one of the operands and also stores result in it. MOVE instruction is used to move the value of variable in register or move the value of register to the memory. For this purpose we used the

registers R1 and R2. In order to solve the expression X = (A + B x C) / (D - E x F), we give the sequence of instructions as follows:

```
MOVE R1, E   ; moves the value of E in R1
MUL R1, F    ; multiplies the value of R1 with variable F MOVE
R2, D  ; moves the value of D in R2
SUB R2, R1   ; subtracts the value of R1 from value of R2 and stores
; the result in R2
MOVE R2, Y   ; moves the value of R2 to a memory location Y MOVE R1,
B  ; moves the value of B in R1
MUL R1, C    ; multiplies the value of C with value in R1
MOVE R2, A   ; moves the value of A in R2
ADD R1, R2   ; adds the values of R1 and R2 and stores the result in
        ; R1
MOVE R2, Y   ; moves the value of variable Y in R2
DIV R1, R2   ; divides the value of R1 with value of R2 and stores in
        ; R1
MOVE R1, X   ; moves the value of R1 as a memory location X
```

d. **3-Address Instructions**: These instructions allow at the most three operands that may be memory locations or registers. The sequence of instructions to compute value of X = (A + B x C) / (D - E x F) is give below:

```
MUL R1, E, F        ; multiply E, F and store in R1
SUB R1, D, R1       ; Subtract contents of R1 from D and store in R1
MUL R2, B, C        ; multiply B, C and store in R2
ADD R2, A, R2       ; Add contents of R2 in A and store in R2
DIV R1, R2, R1      ; Divide contents of R2 by contents of R1 and
; store in R1
MOVE X, R1          ; move the contents of R1 in memory location X
```

Question 14:

Compare zero-, one-, two- and three-address machines by writing programs to compute: X = (A + B x C) / (D - E x F)

For each of the four machines, the instructions available for use are:

0-Address	1-Address	2-Address	3-Address
PUSH M POP M ADD SUB MUL DIV	LOAD M STORE M ADD M SUB M MUL M DIV M	MOVE (X □ Y) ADD (X □ X + Y) SUB (X □ X - Y) MUL (X □ X x Y) DIV (X □ X / Y)	MOVE (X □ Y) ADD (X □ Y + Z) SUB (X □ Y - Z) MUL (X □ Y x Z) DIV (X □ Y / Z)

Answer:

The comparison has been given in the following table with the programs in each column and the total instruction count in the bottom row.

0-Address	1-Address	2-Address	3-Address

PUSH E
PUSH F
MUL
PUSH D
	LOAD E
	MUL F
	STORE s
	LOAD D
		MOVE X, A
		MOVE y, B
		MUL y, C
		ADD X, y
			MUL z, B, C
			ADD w, A, z
			MUL s, E, F
			SUB t, D, s

SU
B PUSH B
PUCH C
MUL
PUSH A
ADD
DIV
POP X
	SUB s
	STORE s
	LOAD B
	MUL C
	STORE t
	LOAD A
	ADD t
	DIV s
	STORE X
		MOVE z, D
		MOVE w, E
		MUL w, F
		SUB z, w
		DIV X, z
			DIV X, w, s

12	13	9	5

b) Assume that M is a 16-bit memory address and that X, Y, and Z are either 16-bit addresses or 4-bit register numbers. The one-address machine uses an accumulator, and the two- and three-address machines have 16 registers and instructions operating on all combinations of memory locations and registers. Assuming 8-bit opcodes and instruction lengths that are multiples of 4 bits, how many bits does each machine need to compute X?

Answer:

0-Address machine: This machine has 8-bit opcode and one memory address of 16 bits. So the instruction length is 8+16=24. The machine needs 24 * 12 = 284 bits to compute X

1-Address machine: This machine has 8-bit opcode and one memory address of 16 bits. So the instruction length is 8+16=24. The machine needs 24 * 13 = 312 bits to compute X

2-Address machine: This machine has 8-bit opcode and two 4-bit register numbers or one 4-bit register and 16 bit memory address. So the instruction length is either 8+4+4=16 or 8+4+16=28. There are 4 MOVE instructions having 28 bits each and 5 arithmetic instructions having 16 bits each. The machine needs 28*4 + 16*5 = 192 bits to compute X

3-Address machine: This machine has 8-bit opcode and three 4-bit register numbers. So the instruction length is 8+4+4+4=20. The machine needs 24 * 5 = 120 bits to compute X

Question 16: In the following code

```
For (i=2 ; i < 100 ; i = i+1)
{
        a[i] = b[i] + a[i]          /*s1*/
        c[i-1] = a[i] + d[i]        /*s2*/
        a[i-1] = 2 * b[i]           /*s3*/
        b[i+1] = 2 * b[i]       /*s4*/
}
```

I. List all dependences (output, anti and true)

II. Indicate whether the true dependences are loop carried or not? III. Why the loop is not parallel?

Solution:

(i)

There are total six dependences in the loop

1. Ther is antidependence from S1 to s1 on a a[i] = b[i] + a[i] /*S1*/
2. There is true dependence form S2 to S1 a[i] = b[i] + a[i] /*S1*/
   c[i-1] = a[i] + d[i] /*s2*/
   Hence, the value of a in S2 is dependet on the result of a in S1.
3. There is loop carried true dependence from S4 to S1 on b.
4. There is loop carried true dependence form S4 to S3 on b.
5. There is loop carried true dependence form S3 to S3 on b.
6. There is loop carried true dependence form S3 to S3 on a.

(ii)

We know that for loop to be parallel, each iteration must be independent of all other. Here in this case, as dependences 3,4,5 (in part i) are true dependences. They cannot be removed by renaming or any such technique. These dependence are loop carried as the iterations of the llp are not independent.

(iii)

The factor mention in (ii), imply the loop is not parallel as the loop is written. Hence, loop can be made parallel by rewriting the loop to find a loop that is functionality equivalent to the original loop that can be made parallel.

Question 17:

The loop given below is a dot product (assuming the running sum in F2 initially 0) and contains a recurrence

```
L.D       F0, 0(R1);          /load X[i]
L.D       F4, 0(R2);          /load Y[i]
MUL.D         F0,F0,F4            /multiply x[i]*y[i]
ADD.D         F2,F0,F2            /add sum = sum +x[i] * y[i]
DADDUI    R1,R1,#-8           /decrement x index i
DADDUI    R2,R2,#-8           /decrement y index i
BNEZ      R1,foo              /loop if not done
```

Assume the pipeline latencies from the table shown below, and a I cycle delayed branch and considering single issue pipelines

Instruction Producing Result	**Instruction Using Result**	**Latency in Clock Cycle**
FP ALU op	Another FP ALU op	3
FP ALU op	Store Double	2
Load double	FP ALU op	1
Load double	Store Double	0

(a) (i) unroll the loop sufficient number of time to schedule it without delay

(ii) show the schedule after eliminating any redundant overhead instruction

(b) Write the unrolled and scheduled code for the transferred code as assume loop body takes 10 cycles

Solution:

(a)

(i) This code has loop carried dependence from iteration I to i+1. It also has high latency dependence within and between loop bodies. Now if we unroll the loop twice in order to avoid any delay, we will get the following result.

```
Foo
        L.D     F0, 0(R1) L.D
        F4, 0(R2) L.D     F6,
        #-8(R1)
        MUL.D   F0, F0, F4   1 from L.D F4, 0(R2) L.D
        F8, #-8(R2)
        DADDUI  R1, R1, #-16
        DADDUI  R1, R1, #-16
        MUL.D   F6, F6, F8   1 from L.D F8, -8(R2) ADD.D
        F2, F0, F2   3 from MUL.D F0, F0, F4
        DADDUI  R2, R2, #-16
        Stall
        BNEZ    R1, Foo
        ADD.D   F2, F6, F2   in slot, and 3 from ADD.D F2, F0, F2
```

Hence, the dependences chain from one ADD.D to the next ADD.D forces the stall

(ii)

In order to unroll further to schedule eliminating the stall (overhead), we take advantage of commutatively and associativity of dot product of two running sums in the loop. One for even elements and one for add elements, and combine the two partial sums outside the loop body.

```
Foo
        L.D     F0, 0(R1)
        L.D     F6, -8(R3) L.D     F4, 0(R2) L.D     F8, -8(R2)
        MUL.D   F0, F0, F4   1 from L.D F4, 0(R2) MUL.D
        F6, F6, F8   1 from L.D F8, -8(R2)
        DADDUI  R1, R1, #-16
        DADDUI  R2, R2, #-16
        ADD.D   R2, F0, F2   3 from MUL.D F0, F0, F4
        BNEZ    R2, Foo
        ADD.D   R2, F0, F2   3 from MUL.D F6, F6, F8
                             And fill the branch delay slot
        ADD.D   F2, F0, F2   combine even and odd elements
```

Conclusion:

Here, the code assumes that the loop executes a non zero, even number of times. The loop itself is stall free, but there are three stalls when the loop exists. The loop body takes 11 clock cycles

B: As loop body takes 10 cycle

Integer Inst.	**FP Inst.**	**Clock Cycle**
Foo		
L.D F0, 0(R1)		1
L.D F6, -8(R1)		2
L.D F4, 0(R2)		3
L.D F8, -8(R2)		4
DADDUI R1, R1, #-16	MUL.D F0, F0, F4	5
DADDUI R2, R2, #-16	MUL.D F6, F6, F8	6
Stall		7
Stall		8
BNEZ R1,Foo	ADD.D F2, F0, F2	9
	ADD.D F2, F0, F2	10
……. …….	…….	…….

Bar: ADD.D F2, F0, F2 14

Question 18:

Consider a code:

```
For ( i = 2 ; i < 100; i+ = 2) A[i]
        = a[50 * i+1]
```

i. Normalize the loop such that start index at 1 and increment it by 1 on every iteration

ii. Write the normalized version of the loop then use GCD test to see if there is dependence.

Solution:

i. By normalize the loop, it leads to a modified c code sa shown below; For (i = 1; i < 50; i++)

```
{
    A[2 * i] = a [(100 * i) + 1]        ;multiple constant by 2
}
```

ii. The GCD test shows the potential for dependences written an array indexed by the function, Ai + b and ci + d only, If the condition (d-b) mod GCD(c, a) =0 is satisfied. Now, applying GCD test, in that case we will get, a=2, b=0, c=100 that allows us to determine dependence in loop. Thus, GCD will be, GCD(2, 100) = 2 and d-b=1. Here, as 1 is factor of 2. Thus, GCD test indicates that there is dependence in the code. In reality, there is no dependence in the code. Since the loop lead it value from a[101], a[201]….a[5001] and again these values to a[2], a[4]…..a[100].

Question 19:

Which approaches increase the amount of ILP?

Solution:

Loop unrolling, software pipelining, trace scheduling and superblocks approaches increase the amount of ILP, which can be exploited by a processor issuing more than one instruction on every clock cycle.

Question 20:

Write down the reasons for which recurrence detection is important

Solution:

Recurrence detection is important for two reasons:

1. Some architecture have special support for executing recurrences
2. Some recurrence can be the source of a reasonable amount of parallelism

Question 21:

Find whether dependence exist in the following loop.

```
For (i = 1 ; i <100; i = i + 1)
{
        X[2 * i + 3] = x[2 *i ] * 5.0;
}
```

Soluion: Here in X[2 * i + 3] such that a =2 and b =3 and in x[2 *i] such that c = 2 and d =0

Thus GCD(a, c) =2 and d-b=-3

Here, as 2 does not divide -3 so no dependence is possible

Question 22:

Define affine index?

Solution:

An affine index is defined as follows;

An array index is affine if it can be written in the form of an expression. Here, a and b are constant, and i is the loop index variable. E.g. in the loop

```
For (i = 1 ; i <100; i = i + 1)
{
        X[2 * i + 3] = x[2 *i ] * 5.0;
}
```

Here, the index value X[2 * i + 3]is affine with a = 2 and b = 3

Question 23:

How compiler finds dependences and its assumption?

Solution:

The compiler detects the dependence using dependence analysis algorithm and this algorithm works on assumptions that:

- Array indices are affine
- There exist GCD of two affine indices

Question 24:

Suppose we are applying the tomasulo‟s algorithm to implement the out of order execution with dynamic scheduling, the tomasulo‟s organization has three load and three store functional units(FU‟s), one “add” FU with three “add” reservation station, and one “multiply” FU with two “multiply” reservation stations. All the load, store, and and multiple FU‟s are pipelined.

A picec of loop code with mostly floating point operations is executed in such a machine as shown in the following figures in cycle1, the instruction “LD F0,)(R1)” is issued. The register R1, used for addressing and iteration control, is 80 in cycle 1. After each iteration, R1 will be deducted by 8 (“SUBI R1, R1, 8”).

The following figure on the right shows the Tomasulo structure with instructions for two iterations except for the loop maintenance instruction (SUBI and BNEZ).

A negative number in “Exec Comp” is used to indicate the remaining cycles for this instruction in the execute stage (e.g, -1 means the next cycle is the execution completion cycle for this instruction).

```
Loop:  L.D      F0, 0(R1)
       MULI.D          F4, F0, F2
       S.D      F4, 0(R1) SUBI
       R1, R1, 8
       BNEZ    R1, Loop
```

The instruction sequence of the loop

The state of Tamasulo‟s organization in cycle 13

a. By the start of cycle 12, how many instructions have completed their issue step?
   List these instructions that are issued before cycle 12
b. At the beginning of cycle 12, what is the next instruction be issued? Can it be issued in cycle 12? Show the instruction and explain the reasons why it can or cannot be issued

Solution:

a. 11 instructions have been issued

```
L.D     F0, 0(r1) MULT.D  F4,
F0, F2
S.D     F4, 0(R1) SUBI
R1, R1, 8
BNEZ    F1, R1, 8
L.D     F0, 0(R1) MULT.D
F4, F0, F2
S.D     F4, 0(R1) SUBI
R1, R1, 8
BNEZ    R1, Loop
L.D     F0, 0(R1)
```

b. MULT.D F4, F0, F2

No. the MULT.D instruction cannot be issued because no multiply reservation station is available. There is a structural hazard.

Question 25:

Suppose number in an array of 200000 randomly generated 16bit unsigned integers. The following code snippet sums up a subset of elements in the array

```
L1      std::sort (numbers, numbers+200000);          //sort the array in ascending order
L2
L3      Long long sum = 0;                            // sum is a 64-bit integer
L4
L5      for(unsigned long j = 0 ; j < 200000 ; j++) {
L6          if (number[i] < 200000) {
L7              sum = sum + numbers[i];
L8          }
L9      }                                             // end of for loop
```

Suppose we compile the code with a moderate optimization. It is known that code in the range L5-L9 runs much more slowly if we remove the code in L1, i.e. , no sorting is done before running code L5-L9 suppose your computer has adopted a 2-bit branch predictor. Please explain why there is a slowdown without sorting.

Solution:

Without sorting, branch predictor has high miss prediction rate for the branch on L6 and hence incurs significant branch penalty. The array numbers contains randomized value, and predictor keeps making inaccurate prediction and switching between taken and not taken states.

Without sorting, the branch for L6 is taken consistently at first and then not taken consistently later when the value in the array is larger than or equal to 30000. The 2-bit predictor only miss predicts a couple of time at the beginning of the loop and when the value in array become equal to or larger than 300000. So very little branch penalty is incurred.

Question 26:

Find the dependences in case of loop level parallelism(LLP) For (i = 1000; i > 0;

```
i = i – I)
        X[i] = X[i] + s ;
```

Solution:

Here, two dependences occur. First, there exist dependence between the two uses of x[i] within the same iteration, so this is loop-level dependency but not loop carried. Second, dependency occurs between the successive uses of i in different iterations, which is loop-carried

Moreover, there exist induction variable x[i]. therefore, dependence can be identified through compiler analysis near source level, and can be eliminated by loop unrolling.

Q.27. We have a single stage, non-pipelined machine and a pipelined machine with 5 pipeline stages. The cycle time of the former is 5 ns and the latter is 1ns.

a. Assuming no stalls, what is the speedup of the pipelined machine over the single stage machine?

SOL: Speed up=5/1=5

b.Given the pipeline stalls 1 cycle for 40% of the instructions, what is the speed up now?

SOL: Speed up = (1 CPI * 5ns) / (1.4 CPI * 1ns) = 3.58

Q.28. For the following, assume that values A, B and C reside in memory. Also assume that instruction operation codes are represented in 8 bits, memory addresses are 64 bits and register addresses are 6 bits.

For each instruction set architecture shown in the table below, how many addresses, or names, appear in each instruction for the code to compute C = A+B and what is the total code size?

Stack	**Accumulator**	**Register-memory)**	**Register (Load-store)**
Push A	Load A	Load R1, A	Load R1, A
Push B	Add B	Add R3, R1, B	Load R2, B
Add	Store C	Store R3, C	Add R3, R1, R2
Pop C			

TABLE: The code sequence for C = A + B for four classes of instruction sets.

SOLUTION:

Stack	**Accumulator**	**Register-memory**	**Register**
Push A(72)	Load A(72)	Load R1, A(80)	Load R1, A(80)
Push B(72)	Add B(72)	Add R2, R1, B(88)	Load R2, B(80)
Add(8)	Store c(72)	Store c, R2(80)	Add R3, R1, R2(80)
Pop c(72)			

Question 15: Choose the correct option:

1. Process with lower CPI"s will always be faster
   a. True
   b. False
2. Instruction count in a program depends on the instruction set architecture used
   a. True
   b. False
3. The Sun SPARC architecture has fewer addressing modes than IA-32(x86)
   a. True
   b. False
4. If we use an instruction in the branch target path to fill a delay slot, we may need to duplicate the instruction.
   a. **True**
   b. False
5. To run on a dynamic scheduling processor, such as an IBM 360/A1 system which implements the Tomasulo algorithm, an ISA mush be 16 or more FP register
   a. Ture
   b. False
6. Which one of the following type of hazards cannot be removed by register renaming?
   a. RAW hazards
   b. War hazards
   c. WAW hazards
   d. None of the above e. All of the above
7. Which of the following factors increases the cost of a processor directly and most significantly?
   a. Supporting Bell Lab"s Unix operating system
   b. Increasing the die area of the processor chip
   c. Increasing the diameter of the wafer d. Increasing the clock cycle time
8. Which one of the following addressing modes are least likely to be used in common programs?
   a. Register direct
   b. Register indirect c. Immediate
   d. Memory indirect
9. With a VLIW design, which of the following component ban be simplified?
   a. Processor
   b. Physical memory (DRAM)
   c. Cache
   d. Complier
10. There are several classes of ISAs-accumulator based ISAs, stack based ISAs, and general purpose register based ISAs. The intel x86 computers (using the IA-32 instruction set) belong to
    a. Accumulator based architecture b. Load/store architecture
    c. General purpose register(GPR) based ISA
    d. Stack based ISA

27-12-15

Q:1-Compare performance of two programs

1	2sec	1.5sec
2	5sec	10 sec

Which computer is faster for each program?

Q:2.We have a single stage, non-pipelined machine and a pipelined machine with 5 pipeline stages. The cycle time of the former is 5 ns and the latter is 1ns.
Assuming no stalls, what is the speedup of the pipelined machine over the single stage machine?

cs704 today at 8:30am 27-12-2015
Q.1: Main advantage of Tomasulo'a design over scoreboead. (5)

Q.2: The MIPS designers wanted the integer multiply and divide instructio...n to operate in parallel with other integer instructions. Since multiply and divide take multiple clock cycle, a valid argument can be: whether it possible to implement precise exceptions or not. (5)
Classify the following statements as: completely accurate, partially accurate, or wrong.

i. It is impossible to implement precise exceptions, since a multiply or divide can raise an exception after instructions that follow it.
ii. It is trivial once the start, and so the timing of all exception is obviously precise… don't remember exactly

Unsolved Questions:

Q.1. (3+3+8=14 Marks)
Assume that we are considering enhancing a machine by adding a vector mode to it. When a computation is run in vector mode it is 20 times faster than the normal mode of execution. We call the percentage of time that could be spent using vector mode the percentage of vectorization. [You don"t need to know anything about how they work to answer this question!] (Repeat = 2)
a. What percentage of vectorization is needed to achieve a speedup of 2?
b. What percentage of vectorization is needed to achieve one-half the maximum?
c. Suppose you have measured the percentage of vectorization for programs to be speed up attainable from using vector mode? 70%. The hardware design group says they can double the speed of the vector rate with a significant additional engineering investment. You wonder whether the compiler crew could increase the use of vector mode as another approach to increasing performance. How much of an increase in the percentage of vectorization (relative to current usage) would you need to obtain the same performance gain? Which investment would you recommend?

Q.2. Consider three branch prediction schemes: branch not taken, predict taken, and dynamic prediction. (5) Assume that they all have zero penalties when they predict correctly and 2 cycles when they are wrong. Assume that the average predict accuracy of the dynamic predictor is 90%. Which predictor is the best choice for the following branches? RR=2

i. A branch that is taken with 95% frequency
ii. A branch that is taken with 70% frequency

Q.3. In a 5-stage pipelined processor, all the instructions are not active in every stage of the pipeline. If we ignore the effects of hazards, which of the following statements are correct? (6)

i. Allowing jumps, branches, and ALU operations to take fewer cycles only helps when no loads or stores are in the pipeline, so the benefits are small.
ii. You cannot make ALU instructions take fewer cycles because of the write back of the result, but branches and jumps can take fewer cycles, so there is some opportunity for improvement.
iii. Instead of trying to make instructions take fewer cycles, we should explore making the pipeline longer, so that instructions take more cycles, but the cycles are shorter. This could improve performance.

Q.4 Describe the following (4+2)
a) List down two advantages and two disadvantages of register-memory architecture.
b) What is the difference between branch and jump instruction?

Q. 5. Answer the following questions (5+5)

a. What is the impact of increasing the size of branch-prediction buffer on two branches in a program?
b. Consider a loop branch that branches nine times in a row, and then is not taken once. What is the prediction accuracy for this branch, assuming the prediction bit for this branch remains in the prediction buffer?

Q. 6. Suppose we have two implementations of same instruction set architecture. Computer A has a clock cycle time of 250 ps and a CPI of 2 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program. Which computer is faster for this program and how much? ET = IC*CPI*CT ETA = IC * 2*250 = 500IC ETB = IC * 1.2 * 500 = 600IC. (7)

Q.8. **For the following sequence of instructions,** list all RAW, WAR and WAW dependencies Repeated = 2 (14)

```
LD R1, 100(R2)      ; load into R1 value at 100+(R2)
ADD R1, R3, R1      ; R1 = R3 + R1
ADDI R2, R3, #-25   ; add -25 to R4 and store in R2
SD 100(R2), R1      ; store R1 at 100+(R2)
ADD R3, R1, R3      ; R3 = R1 + R3
ADDI R1, R1, #25    ; add immediate 25 to R1
```

Use register renaming to eliminate as many dependencies as possible.

Q.10. For the following code sequence: (15)

```
L.D F6, 34(R2)
L.D F2, 45(R3)
MULT.D F0, F2, F4
SUB.D F8, F6, F2
DIV.D F10, F0, F6
ADD.D F6, F8, F2
```

Identify instruction pairs with all the RAW, WAR and WAW hazards.

Example 1: Instruction Flow ADD.D F6,F8,F2 5 10 11-12 14
DIV.D F10,F0,F6 4 13 14-28 29
SUB.D F8,F6,F2 3 6 7-8 9
MUL.D F0,F2,F4 2 6 7-11 12
L.D F2, 45(R3) 1 2 3-4 5 L.D F6, 34(R2) 0 1 2-3 4
Instructions Issue RO Execute Write Back WAR •
Function unit latencies: FPADD = 2 cycles, FPMULT = 5 cycles, FPDIV = 15 cycles, FPLOAD = 2 cycles, Integer = 1 cycle • Cannot read and write a register in the same cycle • All units except FPDIV are pipelined

Q.9. Suppose a program running on a RISC machine performs 16,000,000 instructions during its execution. The total time it takes to execute an instruction is 200 ns, independent of the clock cycle time. The total amount of work that needs to be performed on each instruction is infinitely divisible, so there may be any number of pipeline stages. (15)

a) Complete the table below by computing the stage time, total execution time, and speedup (relative to the non-pipelined case) for the different pipelining depths. Ignore all hazards (i.e., assume ideal pipelining for this part). Neglect stage time increases caused by pipeline register delays, etc., for this part.

Pipeline Depth	Stage time	Total Execution Time	Pipeline Speedup
1	200 ns	3.2 sec	1
2			
4			
8			

b) Now suppose pipelining register delays and processor control overhead adds 10 ns to the latency of each pipeline stage. (So, for example, if there are four pipeline stages, each instruction will have an execution latency of 240 ns and the pipelined machine produces 1 instruction every 60 ns.) What is the maximum speedup that can be obtained through pipelining? Assume there are no hazards (ideal pipelining).

c) Now take into account stalls caused by hazards in the pipeline. Complete the table below using the average stall cycles per instruction listed for each pipeline depth. Ignore stage time increases caused by pipeline register delays, control overhead, etc., for this part.

Pipeline Depth	Average # Stall Cycles/Instruction	Stage Time	Total Execution Time	Pipeline Speedup
1	0.0	200 ns	3.2 sec	1
2	0.6			
4	1.4			
8	4.1			

Q.10. Assume that registers R2, R3 and R4 store initial values of 200, 300 and 400 respectively. Also assume that Mem[200] = 25, Mem[300] = Mem[400] = 50, Mem[500] = 100, Mem[596] = 40, Mem[600] = 200, Mem[700] = 100, Mem[1000] = 150, Mem[1200] = 300, Mem[1600] = 400.
What value will be stored in register R1 and in R2 in each of the following cases: (10 Marks)

a. ADD R1, R2, R3
b. ADD R2, R4, #15
c. ADD R1, R2, (R4)
d. ADD R2, R3, 100(R4)
e. ADD R1, R2, @(600)

[Assume sequential execution with no pipelining. Each instruction is dependent upon the result of the previous instructions.]

	a	b	c	d	e	f	g	h	i	j
R1	500	500	465	465	500	500	625	500	600	1000
R2	200	415	415	400	400	600	600	604	600	600

Q.13. Briefly define the following terms: 10

a. Pipelining
b. Instruction Cycle
c. Finite State Machine
d. Branch predication
e. Dynamic Scheduling

Q.15. Answer the following (7+8)

a. How the presence or absences of control hazard change the pipeline speedup?

Solution: This exercise asks, "How much faster would the machine be . . .," which should make you immediately think speedup. In this case, we are interested in how the presence or absence of control hazards changes the pipeline speedup

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

Solution:

Q.16 In a 5-stages pipeline processor all instructions are not active in every stage of pipeline. If we ignore the effects of hazards, which of following are correct and which are wrong? (5)

a) Allowing jumps, branches & ALU instruction to take fewer stages then the five required by the performance under all circumstances.
b) Trying to allow some instruction to take fewer cycles does not help, since throughput is determined by clock cycles the No. of pipe stages for instruction effects latency not throughput.
c) Allowing jumps, branches and ALU operations to take fewer cycles only helps when no loads or stores are in pipeline so benefits are small.
d) You cannot make ALU instruction take fewer cycles because of write back of results, but branches & jumps can take fewer cycles so there is some opportunity of improvement
e) Instead of trying to make instruction take fewer cycles we should explore making pipeline longer,so that instructions take more cycles ,but cycles are shorter, This could improve performance.

Q.12 Consider a branch-target buffer that has penalties of 0,2 and 2 clock cycles for correct conditional branch predictions in correct prediction and a buffer miss, respectively. Consider a brand-target buffer design that distinguishes conditional and unconditional branches storing the target addresses for a conditional branch and the target instructions for an unconditional branch. RR=2 (15)

a) What is the penalty in clock cycles when an unconditional branch is found in the buffer?
b) Determine the improvement from branch folding for unconditional branches. Assume a 90% hit ratio an unconditional(5marks)

Question No.17. For the following, assume that values A, B and C reside in memory. Also assume that instruction operation codes are represented in 8 bits, memory addresses are 64 bits and register addresses are 6 bits.

a) For each instruction set architecture shown in the table below, how many addresses, or names, appear in each instruction for the code to compute C = A+B and what is the total
b) code size?

Stack	**Accumulator**	**Register (register-memory)**	**Register (load-store)**
Push A	Load A	Load R1, A	Load R1, A
Push B	Add B	Add R3, R1, B	Load R2, B
Add	Store C	Store R3, C	Add R3, R1, R2
Pop C			Store R3, C

TABLE: The code sequence for C = A + B for four classes of instruction sets.

Question No. 18. What is the difference between branch and jump instruction. *15

a. What is the impact of increasing the size of branch-prediction buffer on two branches in a program?
b. Consider a loop branch that branches nine times in a row, and then is not taken once. What is the prediction accuracy for this branch, assuming the prediction bit for this branch remains in the prediction buffer