Solution Of Final Exam Fall 2016

Q1 (a)

(i) Periodic Updates in Benchmarks:

Benchmarking allows to drill down in performance gaps to identify areas for improvement. All standaud benchmarks release new versions periodically because it must be designed to suvive rapid changes in compuēr technology. Therefore, wheu examining benchmark results, pay attention to the version used as, every latest version have some new features added.

(ii) Global Compiler Optimization Techniques: (Techniques across a branch).

1- Global common subexpression elimination.
2- Copy propagation.
3- Code motion → remove code that calculate same value in each iteration.
4- Induction vaiable elimination. simplify/eliminate away addressing calculations within loop.

Q1 (b)

$Speedup_{overall} = 1.75$

$Speedup_{enhanced} = 10$

$Fraction_{enhanced} = ?$

$$S_{overall} = \frac{1}{(1-F_{enh}) + \frac{F_{enh}}{S_{enh}}}$$

$$1.75 = \frac{1}{(1-F_{enh}) + \frac{F_{enh}}{10}} = \frac{10}{10 + F_{enh} - 10F_{enh}}$$

$$\Rightarrow 10 - 9F_{enh} = \frac{10}{1.75} \Rightarrow 10 - 5.714 = 9F_{enh} \Rightarrow F_{enh} = 0.4762$$

$$F_{enh} = 47.62\%$$

Q1 (c)

Methods Of Encoding Branch Conditions in Processor

① Condition Code → Special bit are set by ALU operations, possibly under program control.

② Condition Registers → Tests arbitrary registers with the result of a comparison.

③ Compare & branch → Compare is part of the branch, often compare is limited to subset.

Date:

Q2 (a)

(i) Time of slowest stage = 30 ns.
Time of fastest stage = 25 ns.
Latch delay = 1 ns.
Pipeline clock frequency = ?

$$PCF = \frac{1}{(\text{Time of slowest stage}) + (\text{latch delay})}$$

$$= \frac{1}{(30+1)\,ns}$$

$$= \frac{1}{31\,ns} = 0.032\ GHz.$$

(ii) Latency: No. of intervening cycles b/w an instruction that produces a result and an instruction that uses the result.

Initiation Interval / Repeat Interval: No. of cycles that must elapse b/w issuing two operations of a given type

Q2(b)

Average CPI = ?

Instruction	CPI	Frequency
A	1.7	22%
B	2.1	29%
C	2.7	17%
D	2.4	Remaining. → 32%

Average CPI = (1.7 x.22) + (2.1 x 0.29) + (2.7x0.17) + (2.4x0.32)

= Ans.

Q2 (c)

There are mainly 3 types of data hazards

1) RAW Hazard : (Read after write) (flow/true data dependency)

→ It occurs when I_j tries to read data before I_i writes it

e.g :

I : $R_2 \leftarrow R_1 + R_3$

J : $R_4 \leftarrow R_2 + R_3$

2) WAR Hazard : (Write after Read) (false/Anti data dependence)

→ It occurs when I_j tries to write data

Date:

before Ii reads it -

e.g:

I : $R_2 \leftarrow R_1 + R_3$

J : $R_3 \leftarrow R_4 + R_5$

3) WAW Hazard : (Write after write) (output data dependency)

→ When I_j tries to write output before Ii writes.

e.g:

I : $R_2 \leftarrow R_1 + R_3$

J : $R_2 \leftarrow R_4 + R_5$

Note:

WAR and WAW Hazards occur during the out-of-order execution of instructions.

- Measures to alleviate these hazards:
  - Insert pipeline bubble.
  - Operand forwarding.
  - Use algorithms for out-of-order execution
    i.e : Tomasulo
    Register renaming
    Scoreboard

Q3 (a)

(i) Precise Exceptions

Ability to restart an instruction is called as "precise exception". It refers to, all instructions before fault are committed & those after it can be restarted from scratch.

(ii) Correlating Branch Predictors:

Branch predictors that use the behaviour of other branches to make a prediction are called correlating branch predictors or two-level predictors.

→ Existing correlating predictors add information about the behaviour of most recent branches to decide how to predict a given branch.

(iii) How does Tomasulo's approach of dynamic scheduling perform internal forwarding to resolve RAW hazards?

RAW Hazard is eliminated using internal forwarding as, source operand values that are computed after the registers are read are known by the functional unit or load buffer that will produce them. Results are immediately forwarded to functional units on the common data bus.

Q3 (b)

Tomasulo's Approach Of Dynamic Scheduling

		Issue	Execute	Write Result
LD	$F0, 0(R_1)$	✓	✓	✓
LD	$F6, 0(R_2)$	✓	✓	
MULD	$F4, F0, F2$	✓		
ADD.D	$F8, F4, F6$	✓		
S.D	$F8, 0(R1)$	✓		
L.D	$F0, -8(R_1)$			
L.D	$F6, -8(R2)$			

(To be solved)

	Busy	OP	Vj	Vk	Qj	Qk	A
ADD1							
ADD2							
MUL1							
MUL2							
LOAD1							
LOAD1							
LOAD2							
LOAD3							

Register Status Table

Field	R0	F6	F4	F8
Qi				

Q3 (c)

```
Loop :  L.D      F0, 0(R1)
        ADD.D    F4, F0, F2
        S.D      F4, 0(R1)
        DADDI    R1, R1, 8
        BNE      R1, R2, Loop
```

```
Loop  LD    F0, 0(R1)
      ADDD  F4, F0, F2
      SD    F4, 0(R1)
      DADDI R1, R1, 8
      LD    F0, 8(R1)
      ADDD  F4, F0, F2
      SD    F4, 8(R1)
      DADDI R1, R1, 16
      LD    F0, 16(R1)
      ADDD  F4, F0, F2
      SD    F4, 16(R1)
      DADDI R1, R1, 24
      LD    F0, 24(R1)
      ADDD  F4, F0, F2
      SD    F4, 24(R1)
      DADDI R1, R1, 32
      BNE   R1, R2, LOOP
```

⇒

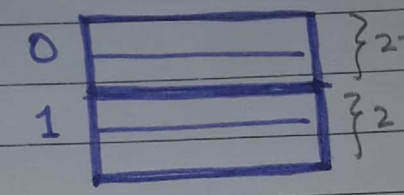
```
Loop  LD    F0, 0(R1)
      LD    F4, 8(R1)
      LD    F8, 16(R1)
      LD    F12, 24(R1)
      ADDD  F3, F0, R2
      ADDD  F6, F4, R2
      ADDD  F7, F8, R2
      ADDD  F9, F12, R2
      SD    F0, 0(R1)
      SD    F4, 8(R1)
      SD    F8, 16(R1)
      SD    F12, 24(R1)

      DADDI R1, R1, 32
      BNE   R1, R2, LOOP.
```

Think Exclusive Think ibex.

Date:

Q4 (a) (i) How does block search take place in set-associative cache?

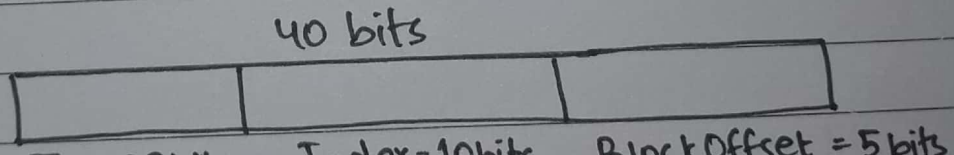
Set-associative cache can be imagined as n*m matrix. The cache is divided into 'n' sets and 'm' cache lines. A memory block is first mapped onto a set and then into cache line and searched in the same way/order.



(ii) Memory refereuces = 1000.
L1 misses = 40
L2 misses = 20.

	LMR	GMR
L1	40/1000	40/1000
L2	20/40	20/1000

Q4 (b)



(i)

Physical Address = Tag + Index + BO
= 25 + 10 + 5 = 4 bits

Size of MM = 2^{40} bytes.

(Index) (BO)

Cache Size = no. of block x blockSize

$= 2^{10} \times 2^5$

Cache size $= 2^{15}$ bytes

(ii)

BlockSize → block offset bits

BlockSize = 2^5 bytes.

No. of blocks → $2^{\text{index bits}}$.

No. of blocks = 2^{10} bytes

(iii) In 4-way Set Associative Cache

$$\text{No of sets} = \frac{\text{No. of blocks}}{\text{Blocks per set}} = \frac{2^{10}}{2^2} = 2^{8}$$

8 → index bits

Tag = 27 bits	I = 8 bits	BO = 5 bits

Q4 (c)

Categories of cache miss:

1. Compulsory Miss → First request to cache block leads to it.

   Measure: large block reduces compulsory miss but increase other types of misses

2. Capacity miss → Occur when cache is too

 Date: _____

small and blocks are to be discarded.

Measure : Large cache memory reduces the miss.

③ Conflict Miss → Occur when several blocks are mapped to same set or block frame.

Measure : Fully associative placement avoid all conflict misses.

Q5 (a) i Advantage of write-back over write-though

Write-though → Write is done synchronously both to the cache and to backing store.

Write-back → Write is done only to cache. Modified cache block is written back to the store, just before it's replaced.

* Advantage of write-back over write though is that, when data is updated it's only written to the cache. So, this mode has a faster data write speed as compared to write-though policy

(ii) 4C2ABC95

Cache Size = 128K bytes. = $2^7 \cdot 2^{10}$ bytes = 2^{17} bytes.

Block Size = 32 bybe = 2^5 bytes. → block offset.

No of blocks = $\frac{\text{Cache size}}{\text{block size}} = \frac{2^{17}}{2^5} = 2^{12}$ → Index

Tag bits = 32 - 12 - 5 = 15 bits.

32 → Physical address bits.

A 10
B 11
C 12
D

Date:

4C2ABC95 : 0100 1100 0010 1010 1011 1100 1000 1010 1

Tag | Index | Block offset

Reducing miss penalty

Q5 (b) Explain, (i) Multilevel Cache (ii) Non-blocking cache → increasing cache band width

Reducing miss penalty ← (iii) Critical word first (iv) Hardware prefetching, techniques for cache optimization, identifying component of memory access time which is improved?

(In notes)

Q5 (c) i Fast Page Mode Operations In DRAM:

FPM DRAM allows faster access to data in same row or page. Page mode memory works by eliminating the need for a row address if data is located in row previously accessed.

(ii) Write back with modified bit.

Q6 (a) (i) Flynn's Classification Scheme

1. SISD → single instruction, single data
2. SIMD → " " , multiple "
3. MISM → Multiple " , single "
4. MIMD → " " , multiple "

(ii) Software models to exploit TLP:

1. Parallel processing → execution of a tightly coupled set of threads collaborating on single task.

On a single task.

ibex.

(2) Request level parallelism → execution of multiple, relatively independent processes that may originate from one or more users. (may originate from one or more users.)

(iii) UMA vs. NUMA (Also called symmetric multiprocessors)

- UMA (Uniform Memory Access) → all processors have same latency to access memory. This architecture is scalable only for limited number of processors.
- NUMA (Non-Uniform Memory Access) → each processor has it's own local memory, memory of other processor is accessible but latency to access them is not same.
  ↓ Also called as distributed shared memory.

Q6 (b) Challenges of Parallel Processing:

(1) Limited parallelism available in programs → It gets difficult to achieve good speedup in // processors.

Solution: a) Develop new algorithms having better // performance.
b) Software development efforts should be geared toward pallelism & parallelizing compilers.

(2) Large latency of remote access in a parallel processor → Introduced due to the high cost of communication.

Solution: a) Reduce frequency of remote access by hardware mechanism i.e: caching of shared data
b) Software mechanism i.e prefetching or multithreading.

ibex.

Date: _____

Q6 (c) 2016.

RAID (Redundant Array of Independent Disks)

→ It's a way of storing data in different places on multiple hard disks to increase performance or provide fault tolerance or both.

(i) How dependability of RAID is improved?

It can be improved by putting multiple harddrives together. Because use of multiple disks increase (MTBF) Mean Time B/w Failure, and also increase fault tolerance.

(ii) RAID4 vs RAID5

Raid4 and Raid5 both uses block striping. The major difference is in parity data which is distributed across all drives in in RAID array rather than on disk. This makes RAID5 more protected against data loss. Raid5 is faster than Raid4 because there is no single parity disk that will create a data input bottleneck. In Raid4 array, array can only write as fast as parity dick.