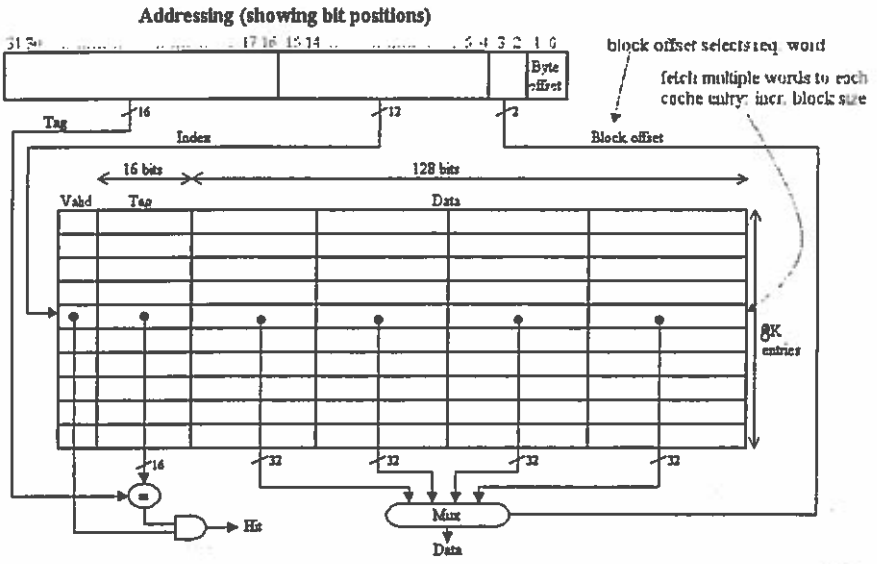
Department of Computing Examinations — 2013–2014 Session	*Confidential*
MODEL ANSWER and MARKING SCHEME	
First Examiner WL	Paper Code 210 = EE2.13
Second Examiner dt10	Question 1 Page 1 *out of* 1
Question labels in left margin	Mark allocations in right margin

1a adv: simple, fast
disadv: rigid mapping, not exploit spatial locality — 2

b total number of words = $\frac{\alpha}{\gamma}$ = total number of blocks
tag size = $\beta - (\log\alpha - \log\gamma) - \log\gamma = \beta - \log\alpha$
total size = num of words × (data size + tag size + valid bit)
$= \frac{\alpha}{\gamma}(8\gamma + (\beta - \log\alpha) + 1) = \frac{\alpha}{\gamma}(8\gamma + \beta + 1 - \log\alpha)$ — 5

c adv: fetch multiple words per block : exploit spatial locality
disadv: complex, slower, more complex control for write hit/miss — 2

d — 6



e total number of blocks = $\frac{\alpha}{\gamma w}$ — 5
tag size = $\beta - \log\alpha$
total cache size = $\frac{\alpha}{\gamma w}(8\gamma w + \beta + 1 - \log\alpha)$
when $w = 1$, get same result as Part b

2a i) Give two examples, with justification, where the MIPS instruction set simplifies compiler design and implementation, compared to a CISC ISA.

* *The large number of available registers to make register allocation simpler.*
* *The uniform treatment of registers, allowing them to be used as both addresses or data.*
* *The lack of architecture state such as flags.*

Marks: 3

ii) Give two examples where the MIPS instruction set has been balanced towards performance or architectural simplicity, at the expense of compiler or programmer convenience.

* *The inability to load words at unaligned addresses.*
* *Cannot load a full 32-bit constant in one instruction.*
* *The separate multiply and divide registers.*
* *Can only use a 16-bit pointer offset.*

Marks: 3

b The following code simulates a very simple processor with five instructions:

```
1  unsigned IM[65536], DM[65536];
2  unsigned pc=0, acc=0;
3
4  while(1){
5    unsigned instr=IM[pc];
6    pc=pc+1;
7
8    unsigned opcode=instr>>16, arg=instr&0xFFFF; // opcode=top 16 MSBs, argument=16 LSBs
9
10   if(opcode==0){
11     DM[arg]=acc;
12   }else if(opcode==1){
13     acc=DM[arg];
14   }else if(opcode==2){
15     acc=arg;
16   }else if(opcode==3){
17     acc=acc+DM[arg];
18   }else if(opcode==4){
19     acc=acc-DM[arg];
20   }
```

```
21 }
```

i) Describe the effect of the following two instructions: 0x2FFFF; 0x8000.

The first instruction sets the accumulator to 0xFFFF. The second writes it to address 0x8000, so now DM[0x8000]==0xFFFF.

Marks: 3

ii) What is the common name for this type of instruction set architecture?

It is an accumulator architecture.

Marks: 1

iii) Suggest names and an assembly-style format for the five instructions.

A) *write [ADDR]*
B) *read [ADDR]*
C) *load IMM*
D) *add [ADDR]*
E) *sub [ADDR]*

Marks: 3

iv) Give a minimal length sequence of instructions (in your assembly format) which reads the value at address 0x2000, multiplies it by 15, and stores it to address 0x2000. If necessary, use address 0x4000 as temporary storage.

A) *read [0x2000]*
B) *add [0x2000]*
C) *write [0x4000]*
D) *add [0x4000]*
E) *write [0x4000]*
F) *add [0x4000]*
G) *write [0x4000]*
H) *add [0x4000]*
I) *sub [0x2000]*
J) *write [0x2000]*

Marks: 4

v) Suggest a single instruction, along with code to add to the simulator, which would add data-dependent conditional branching to the processor.

The new instruction could be branch if accumulator not zero, with the implementation:

```
1 // bne IMM
2
3 }else if(opcode==5){
4   if(acc!=0)
5     pc=arg;
6 }
```

Marks: 3

The two parts carry, respectively, 30%, and 70% of the marks.