Computer Architecture EE2-13

Department of Computing Examinations — 2015–2016 Session *Confidential*

MODEL ANSWER and MARKING SCHEME

Examiner wl Paper Code C210

Question 1 Page 1 *out of* 1

Question labels in left margin — Mark allocations in right margin

1a $m \times 9 = 8m + m$, so

```
sll $2, $1, 3
add $2, $1, $2
```

Take 3+ cycles, $5/3 = 1.67$ times faster than the 'mult' instruction — 6

b If $k = 1022 = \underbrace{1111111110}_{9\text{ '1's}}$, need 9 shifts and 8 adds

Let $1 contains the number m, $2 contains the result 9m, $3 provides temporary storage

```
    sll  $2 $1  1
    sll  $3 $1  2
1   add  $2 $2  $3
    sll  $3 $1  3
2   add  $2 $2  $3
    sll  $3 $1  4
3   add  $2 $2  $3
    sll  $3 $1  5
4   add  $2 $2  $3
    sll  $3 $1  6
5   add  $2 $2  $3
    sll  $3 $1  7
6   add  $2 $2  $3
    sll  $3 $1  8
7   add  $2 $2  $3
    sll  $3 $1  9
8   add  $2 $2  $3
```

9 shifts and 8 adds

This takes $9n + 8(2n) = 25n$,

5 times slower than 'mult' instruction — 6

c Using Booth's algorithm, replace the 8 adds by 1 subtract, since

$m \times 1022 = m(1024 - 2) = 1024m - 2m$

Multiply by 1024 is shift left by 10 bits, so

```
sll  $2 $1  2
sll  $3 $1  10
subu $2 $3 $2
```

This takes 4n cycles, $\frac{5}{4}$ times faster than the mult instruction — 8

Q2

Department of Computing Examinations – 2015 - 2016 Session		Confidential
MODEL ANSWERS and MARKING SCHEME		
First Examiner: **David Thomas**		Second Examiner: **Wayne Luk**
Paper: **C210=E2.13 - Computer Architecture**	Question: **2**	Page **1** of **3**

2a i) Considering L1 vs L3 cache levels, describe two properties of caches and how you would expect them to vary between L1 and L3.

* *The capacity of the L1 cache will be smaller than the L3.*
* *The hit time of the L1 cache will be lower than the L3.*
* *The associativity of the L1 cache will be lower than the L3.*

Marks: 2

ii) Give a fragment of code that demonstrates both temporal locality and spatial locality. You can use any language as long as semantics are clear.

```
uint32_t *pX = ...;
uint32_t acc=0;
for(int i=0;i<100;i++){
  // Loops over sequential words (spatial)
  // Visits each word 10 times (temporal)
  acc=acc+pX[i%10];
}
```

Marks: 3

iii) Assume a 2-way set associative cache with `w` bytes per word, `b` words per block, and a total cache size of `c` bytes. Given a `byte_address`, give statements for calculating the `set_index`. Use intermediate variables that show your working.

```
word_address = byte_address / w;
block_address = word_address / b;
num_sets = c / (w*b*2);
set_index = block_address % num_sets;
```

Marks: 4

b A CPU designer has suggested that the MIPS ISA is extended with an integer multiply-accumulate instruction, with the following syntax and semantics:

```
mul_add $a, $b, $c // a = a + b * c
```

where `$a`, `$b`, and `$c` are all registers.

i) How many data operand registers does a MIPS R-type instruction support?

Three (1 destination; 2 source).

Marks: 2

ii) How would the register-file of a 5-stage pipelined MIPS need to be modified in order to support this instruction?

At the moment the register-file only supports the reading of two operands per cycle. This instruction needs to read three operands, so another read port would need to be added.

Marks: 2

iii) Give a sequence of standard MIPS instructions that is equivalent to `mul_add`. You may use the register `$t` as a temporary register.

```
mulu $b, $c
nop          // Needed according to MIPS ISA
mflo $t
add $a, $a, $t
```

Marks: 3

iv) Assume that both the standard MIPS and extended MIPS execute one instruction per cycle, and caches are warm. Provide a quantitative estimate for the speedup that `mul_add` could provide for the following function. State any assumptions, with a brief explanation of why they are reasonable.

```
uint32_t dot_product(uint32_t n, uint32_t *a, uint32_t *b)
{
  uint32_t acc=0;
  for(uint32_t i=0; i<n; i++){
    acc = acc + a[i] * b[i];
  }
  return acc;
}
```

The inner loop of the function must contain:

* *Two loads*
* *A multiply accumulate*
* *Two loop increments*
* *A branch.*

My assumptions (others could be acceptable, e.g. loop pipelining) are:

* *The compiler can avoid load-use stalls, for example by loop skewing.*
* *No loop pipelining is used, as the compiler is optimising for code size.*
* *One instruction can be placed in the delay slot, as there are a couple of increments.*
* *For the standard MIPS, I assume one instruction can be placed in the NOP.*

This results in standard MIPS = 2+3+2+1=8, and extended MIPS = 2+1+2+1=6. So asymptotically an estimate of speed-up is 8/6 = 4/3 = 1.25x.

Marks: 4

The two parts carry, respectively, 45%, and 55% of the marks.