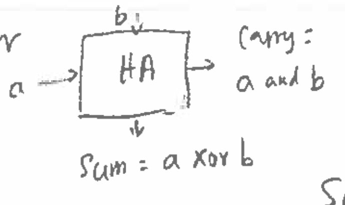
Department of Computing Examinations — 2014–2015 Session *Confidential until 1 July 2015*

MODEL ANSWER and MARKING SCHEME

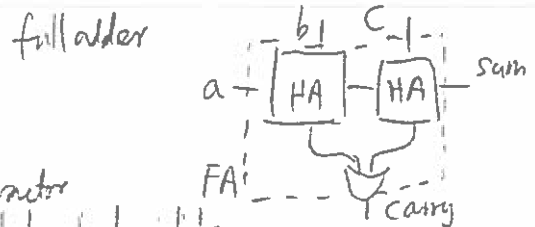
Examiner	wl	Paper Code	C210 / EE2-13
	Question 1	Page 1	out of 1

Question labels in left margin — Mark allocations in right margin

1 a halfadder

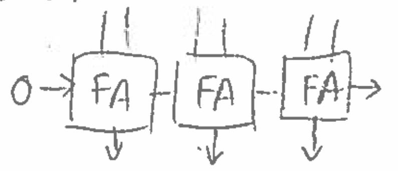


full adder

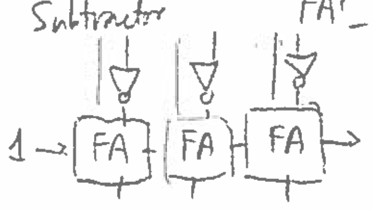


2

b adder



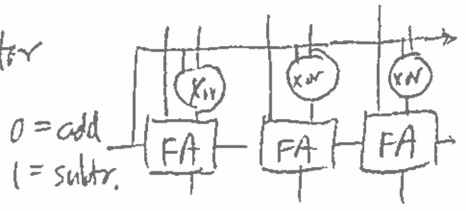
Subtractor



4

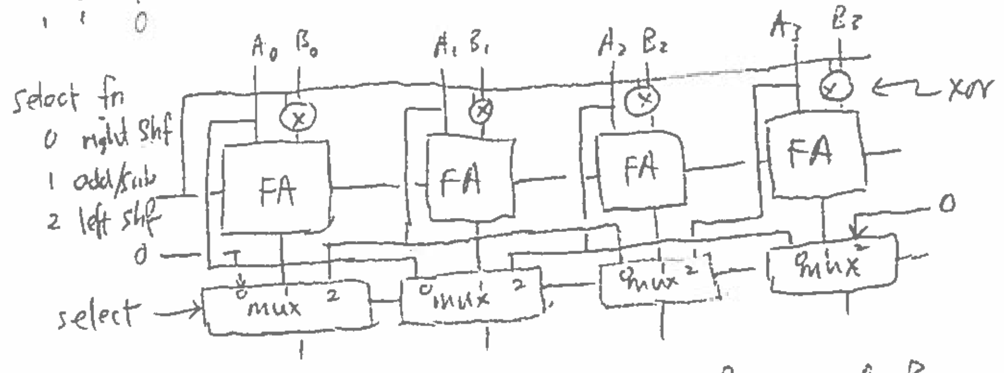
c programmable adder/subtractor

0	0	xor
0	1	0
1	0	1
1	1	0



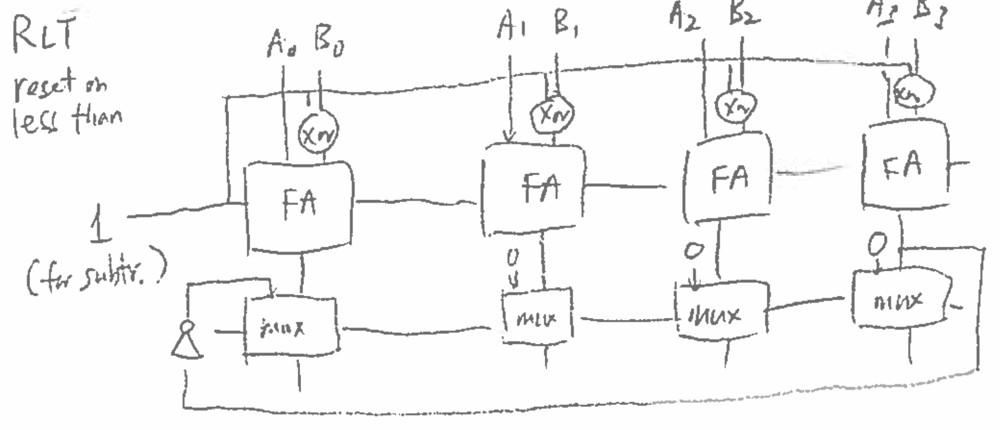
4

d



5

e



5

ALL BOOKWORK

2 The following function performs a very weak hash of an array of data:

```
unsigned hash(int N, const uint32_t *P, int K, int M)
{
  // Pre—condition: 1 <= K < N
  unsigned acc=0, offset=0;
  for(int i=0; i<N; i++){
    unsigned x = P[offset];   // Read next word
    acc = (acc/2) + M*x;     // Fold it into the running hash
    offset = (offset + K) % N;   // Wrap around offset modulo N
  }
  return acc;
}
```

This code is to be compiled for and executed in a standard 5-stage MIPS architecture.

a Describe what event will happen if the function is called with byte address P=4098, and how it will be handled. BOOKWORK

The compiler will issue a lw instruction, but the address is not aligned. This will raise an exception, so the processor will save the PC and the cause into the exception registers, then jump to the appropriate exception handler in the interrupt table.

Marks: 3

b Identify a potential load-use data hazard in the code, and explain how a compiler would avoid it. BOOKWORK

The output of the load at the start of the loop could be consumed in the very next instruction. The compiler would re-order the instructions to make sure that acc/2 happened first.

Marks: 3

c Assume an L1 data-cache with 8 blocks containing 16-bytes each.

i) For a direct-mapped cache, give the sequence of block addresses and block indices accessed during hash for N=8, P=4100 (byte address), K=1.

A) *4100: 256, 0*

B) *4104: 256, 0*

C) *4108: 256, 0*

D) *4112: 257, 1*

RELATED TO COURSEWORK

E) *4116: 257, 1*

F) *4120: 257, 1*

G) *4124: 257, 1*

H) *4128: 258, 2*

Marks: 3

ii) For an initially empty LRU fully-associative cache, how many bytes of data will be fetched into L1 during hash N=1024, byte address P=1024, K=8? Comment on the total bytes fetched into the cache, versus the total size of the P array, versus the total data used by the program.

*The hash will read byte offsets 1024, 1024+8*4, 1024+8*4*2, 1024+8*4*3..., wrapping around after 1024/8=128 iterations. Each block will read in 16 bytes, for a total of 16*128=2048 bytes. There are only 8 blocks in the cache, so all blocks will be evicted before the next pass through. It takes 1024/128=8 iterations to get through the block, so the total bytes read is 16384.*

There are 16384 bytes read, for a total of 4096 bytes in the P array, and a total of 4096 bytes used in the program, so the data access pattern is causing a lot of inefficiency. APPLICATION OF KNOWLEDGE ANALYSIS

Marks: 4

d i) Describe how multiplication and division instructions specify the destination register, and explain why they do not use the same approach as other ALU instructions. BOOK WORK

Both division and multiplication produce two arguments into implicit registers, rather than a single explicit destination. They are also slower, possibly requiring multiple cycles. For both reasons, it makes sense to put them in special registers which are accessed in a different way, in order to ensure they do not interfere with the speed and flow of the more common ALU instructions.

Marks: 3

ii) For the statements involving division, show how each can be expressed without a division instruction, and whether it would be faster or not in a 5-stage MIPS. APPLICATION OF KNOWLEDGE + ANALYSIS

The division by two on line 7 can safely be replaced by a right shift. This is one instruction compared to two, so will always be faster in any MIPS architecture.

The modulo N:

A) *offset=offset+K*

B) *divu(offset, N)*

C) *offset=mflo()*

could be replaced by:

A) *offset=offset+(K-N); // Original code had offset+K*

B) *bltz done*

C) *offset=offset+N; // Fast case in delay slot: didn't need to wrap*

D) *offset=offset-N; // Slow-case: did need to wrap*

E) *done: (original code)*

In the fast case it takes two instructions, which is the absolute minimum a divu/mfhi would take, and we know that div is usually much slower. In the worst case it will take three instructions, which is likely to be at least as good as a typical iterative divider.

Marks: 4

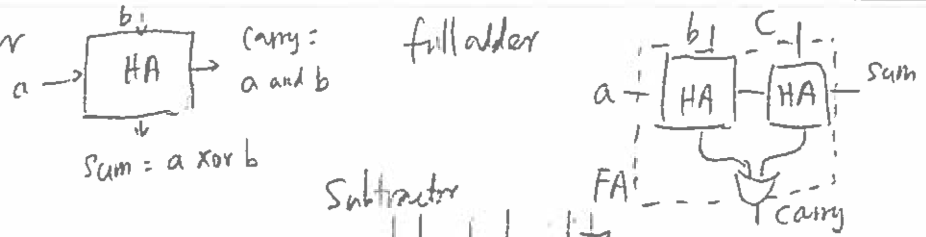
The four parts carry, respectively, 15%, 15%, 35%, and 35% of the marks.

MODEL ANSWER and MARKING SCHEME

Examiner	wl	Paper Code	C210
		Question 1	Page 1 *out of* 1

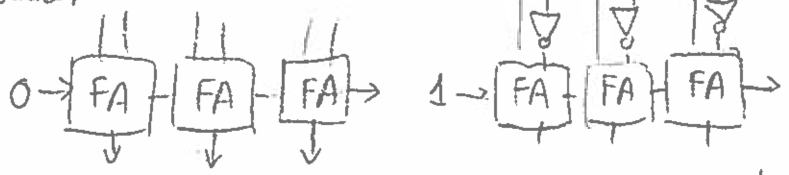
Question labels in left margin — Mark allocations in right margin

1 a halfadder — fulladder



2

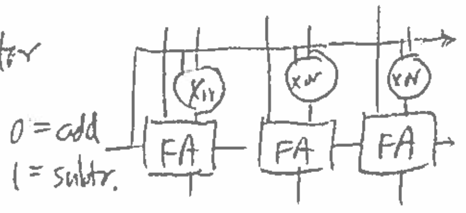
b adder — Subtractor



4

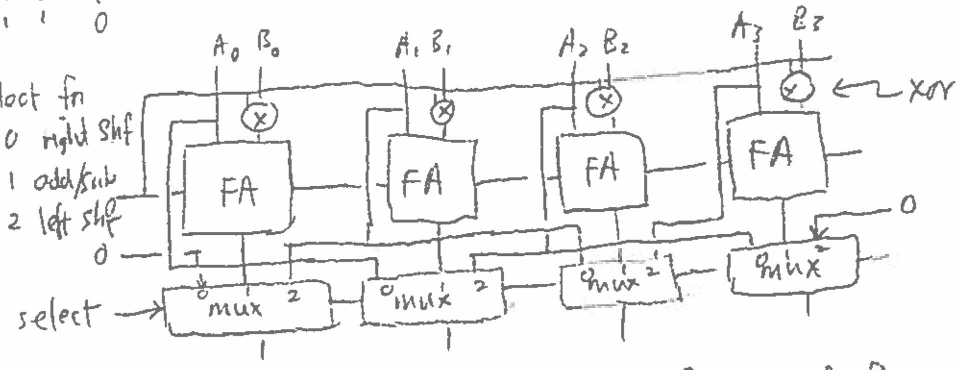
c programmable adder/subtractor

u	v	xor
0	0	0
0	1	1
1	0	1
1	1	0



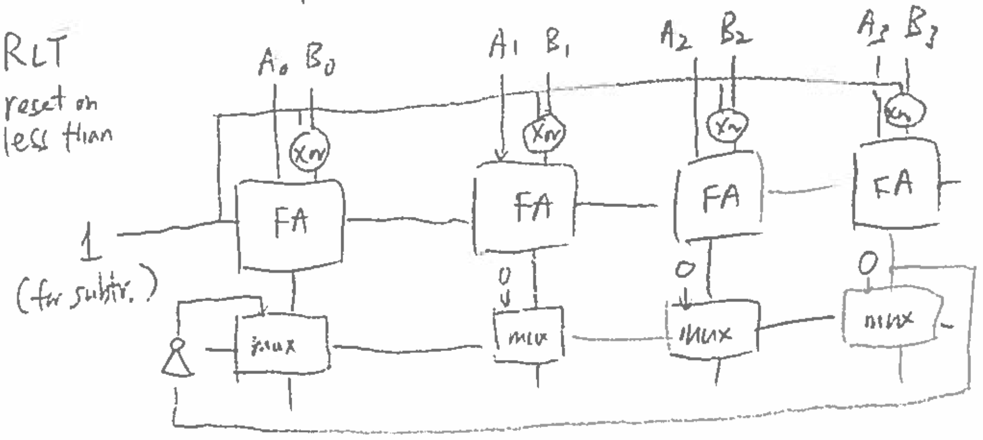
4

d select fn
0 right shf
1 add/sub
2 left shf



5

e RLT
reset on less than



5

2 The following function performs a very weak hash of an array of data:

```
unsigned hash(int N, const uint32_t *P, int K, int M)
{
  // Pre—condition: 1 <= K < N
  unsigned acc=0, offset=0;
  for(int i=0; i<N; i++){
    unsigned x = P[offset];   // Read next word
    acc = (acc/2) + M*x;      // Fold it into the running hash
    offset = (offset + K) % N;   // Wrap around offset modulo N
  }
  return acc;
}
```

This code is to be compiled for and executed in a standard 5-stage MIPS architecture.

a Describe what event will happen if the function is called with byte address P=4098, and how it will be handled.

The compiler will issue a lw instruction, but the address is not aligned. This will raise an exception, so the processor will save the PC and the cause into the exception registers, then jump to the appropriate exception handler in the interrupt table.

Marks: **3**

b Identify a potential load-use data hazard in the code, and explain how a compiler would avoid it.

The output of the load at the start of the loop could be consumed in the very next instruction. The compiler would re-order the instructions to make sure that acc/2 happened first.

Marks: **3**

c Assume an L1 data-cache with 8 blocks containing 16-bytes each.

i) For a direct-mapped cache, give the sequence of block addresses and block indices accessed during hash for N=8, P=4100 (byte address), K=1.

A) *4100: 256, 0*

B) *4104: 256, 0*

C) *4108: 256, 0*

D) *4112: 257, 1*

E) *4116: 257, 1*

F) *4120: 257, 1*

G) *4124: 257, 1*

H) *4128: 258, 2*

Marks: 3

ii) For an initially empty LRU fully-associative cache, how many bytes of data will be fetched into L1 during hash N=1024, byte address P=1024, K=8? Comment on the total bytes fetched into the cache, versus the total size of the P array, versus the total data used by the program.

*The hash will read byte offsets 1024, 1024+8*4, 1024+8*4*2, 1024+8*4*3..., wrapping around after 1024/8=128 iterations. Each block will read in 16 bytes, for a total of 16*128=2048 bytes. There are only 8 blocks in the cache, so all blocks will be evicted before the next pass through. It takes 1024/128=8 iterations to get through the block, so the total bytes read is 16384.*

There are 16384 bytes read, for a total of 4096 bytes in the P array, and a total of 4096 bytes used in the program, so the data access pattern is causing a lot of inefficiency.

Marks: 4

d i) Describe how multiplication and division instructions specify the destination register, and explain why they do not use the same approach as other ALU instructions.

Both division and multiplication produce two arguments into implicit registers, rather than a single explicit destination. They are also slower, possibly requiring multiple cycles. For both reasons, it makes sense to put them in special registers which are accessed in a different way, in order to ensure they do not interfere with the speed and flow of the more common ALU instructions.

Marks: 3

ii) For the statements involving division, show how each can be expressed without a division instruction, and whether it would be faster or not in a 5-stage MIPS.

The division by two on line 7 can safely be replaced by a right shift. This is one instruction compared to two, so will always be faster in any MIPS architecture.

The modulo N:

A) *offset=offset+K*

B) *divu(offset, N)*

C) *offset=mflo()*

could be replaced by:

A) *offset=offset+(K-N); // Original code had offset+K*

B) *bltz done*

C) *offset=offset+N; // Fast case in delay slot: didn't need to wrap*

D) *offset=offset-N; // Slow-case: did need to wrap*

E) *done: (original code)*

In the fast case it takes two instructions, which is the absolute minimum a divu/mfhi would take, and we know that div is usually much slower. In the worst case it will take three instructions, which is likely to be at least as good as a typical iterative divider.

Marks: 4

The four parts carry, respectively, 15%, 15%, 35%, and 35% of the marks.