NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

EXAMINATION FOR

Semester 1 AY2009/10

CS2100 – COMPUTER ORGANISATION

Nov 2009 **Time allowed: 2 hours**

Your Matriculation Number:

INSTRUCTIONS TO CANDIDATES

1. This examination paper consists of **SIX (6)** questions and comprises **EIGHTEEN (16)** printed pages including this page.
2. This is an **OPEN BOOK** examination. You may use any approved calculators but not any PDA or laptop, especially those capable of external connectivity or communication.
3. Answer *all* questions. Note that the full mark for each question is different.
4. Write your answers on *this* **QUESTION AND ANSWER SCRIPT**. Answer only in the space given. Any writing outside this space will not be considered. No other submission is allowed.
5. Fill in your Matriculation Number with a pen, clearly on every page of this QUESTION AND ANSWER SCRIPT.
6. You may use pencil to write your answers.
7. At the end of the examination, please check to ensure that your script has all the pages properly stapled together.
8. Note that when a number is written as "0xNNNN" it means that "NNNN" is in base 16.

Total Score _____ /100

QUESTION 1 (20 marks)

(1a) Convert **-125.5** into IEEE Standard 754 single precision binary floating point number, leaving your answer as an 8-hexadecimal digit number. (2 marks)

ANSWER:

0xc2fb0000

(1b) What is the result (in decimal) of interpreting the hexadecimal number **`0x3d400000`** as a IEEE Standard 754 single precision binary floating point number? (2 marks)

ANSWER:

0.046875

(1c) Convert the number in (1b) into IEEE 754 *double* precision binary floating point number, giving your result as a hexadecimal string. (4 marks)

ANSWER:

0x3fa8000000000000

(1d) A comparison of two positive normalized numbers in the IEEE Standard 754 format will yield the same result as treating the bits as integers and comparing them. For example, if X is 0.5 and Y is 0.25, then the IEEE representation for X and Y is 0x3f000000, and 0x3e800000, respectively. As floating point numbers, X > Y. However, when taken as integers, we also have 0x3f000000 > 0x3e800000. This works for any precision. Show why this is so. (8 marks)

ANSWER:

If the two numbers are of opposite signs, then trivially, the results of comparing them as floating point or as integer will be the same as it will be based on the sign of the numbers.

If the two numbers are positive and has the same exponent, *e*, then the comparison will be down to the fractional part. Let f(X) be the normalized floating point number represented by X. f(X) = 1.fraction(X) × 2^e and f(Y) = 1.fraction(Y) × 2^e then f(X) > f(Y) if and only if fraction(X) > fraction(Y). This implies I(X) = fraction(X) + *c* > I(Y) = fraction(Y) + *c* where *c* is the constant obtained by the exponent bits shifted left to the position of the exponent, and I(X) is X interpreted as a signed-magnitude number.

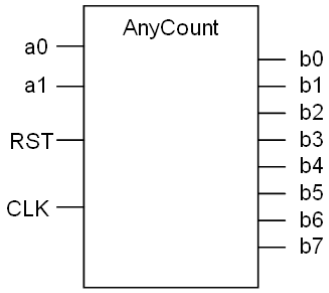
If the two numbers are positive and have different exponents, let us assume, without loss of generality, the true_exponent(f(X)) > true_exponent(f(Y)). Now exponent(X), i.e. the exponent in the representation of f(X) is true_exponent(f(X)) + bias. This implies that exponent(X) = true_exponent(f(X)) + bias > true_exponent(f(Y)) + bias = exponent(Y). This in turn implies I(X) = exponent(X) × 2^i + fraction(X) > exponent(X) × 2^i > exponent(Y) × 2^i + fraction(Y) = I(Y), where *i* is the bit position of the start of the exponent and fraction() is at most *i*-1 bits.

(1e) What if the two normalized numbers are negative? (4 marks)

ANSWER:

Interpreting the numbers in signed-magnitude representation will yield the desired effect.

QUESTION 2 (15 marks)
Consider the following circuit called AnyCount:



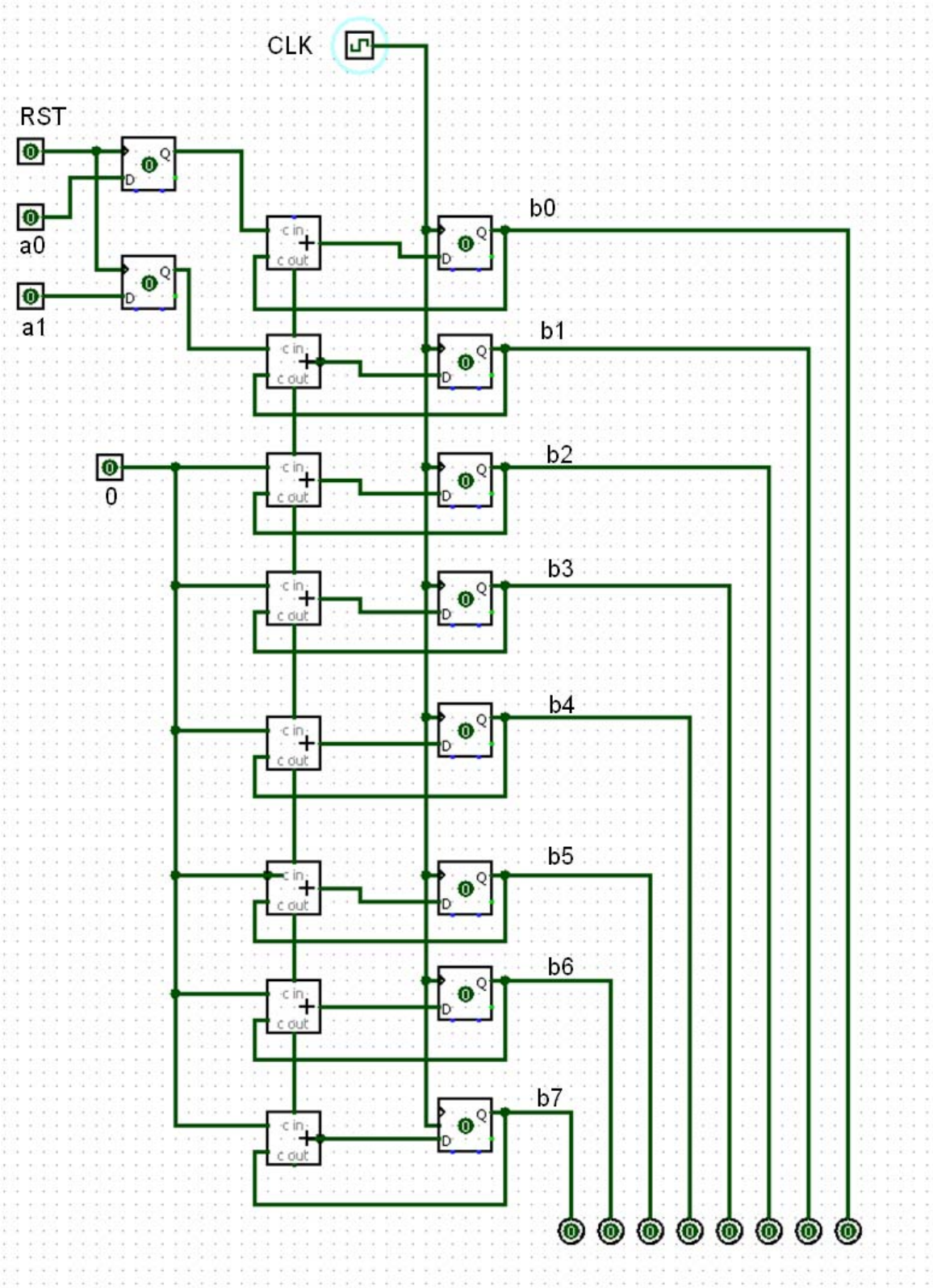
It operates as follows: the user first gives the two bit number a = a1 a0, and then assert the RST line. Once that is done, the circuit will output a, 2a, 3a, etc. (modulo 16) at each clock cycle on its 8-bit output b = b7 b6 b5 b4 b3 b2 b1 b0. The initial output (before a is given) is zero. Furthermore, at anytime, the user may change the value of a to say a'. If that happens, at the next cycle, the output will be a', then 2a' at the next clock cycle, and so on.

(2a) Using full adders, flip flops, latches, and any logic gate you need, show how AnyCount can be implemented. (10 marks)

ANSWER:

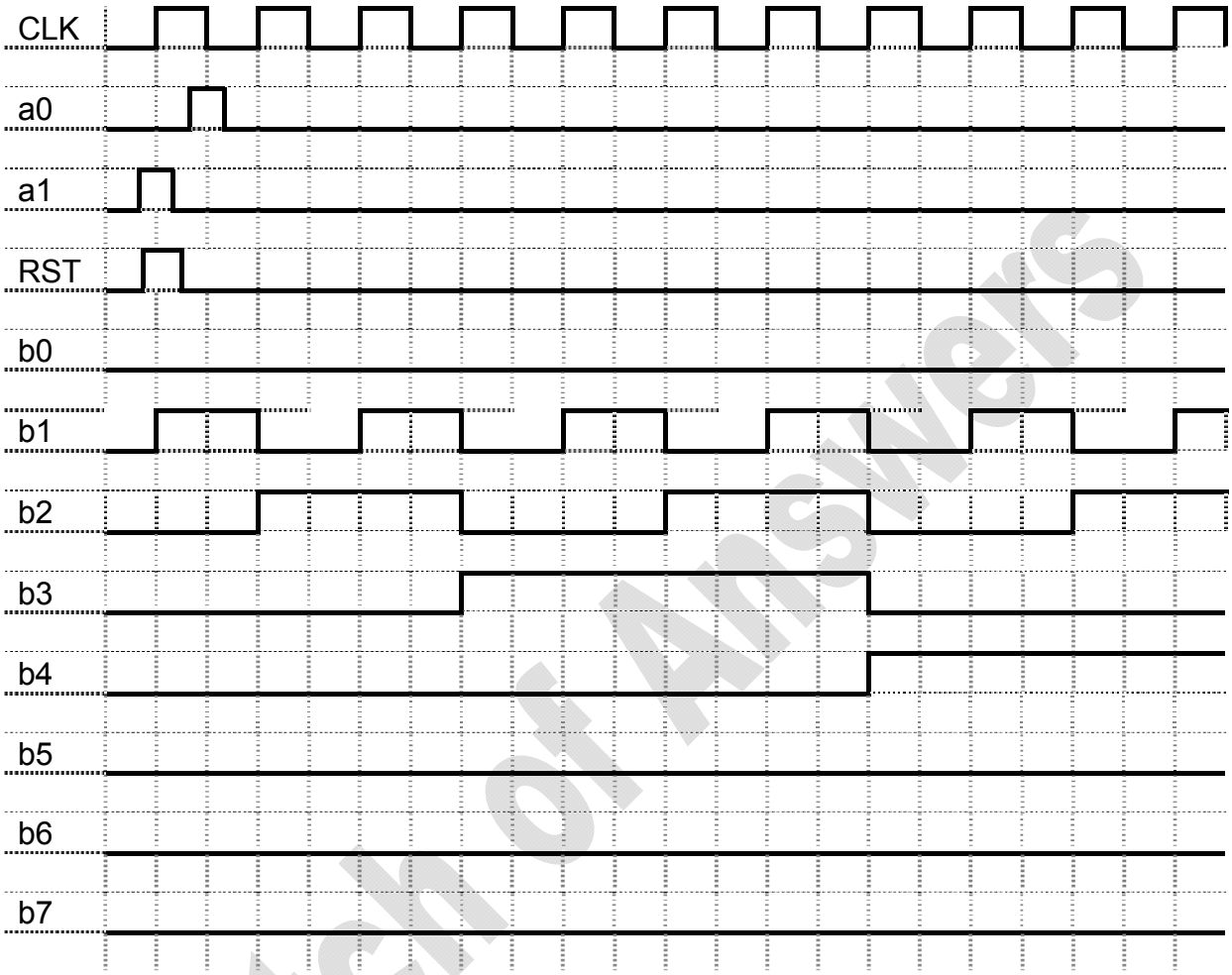
See next page.

Answer for (2a) cont'd:



(2b) Assuming all flip-flops are positive edge-triggered, complete the timing diagram below. (5 marks)

ANSWER:



QUESTION 3 (20 marks)

(3a) The C string function **`strchr(str, c)`** searches the NULL-terminated C string **`str`** (i.e., an array of characters terminated by an element that holds the value 0) for the first occurrence of the character **`c`** and returns the address of that first occurrence. Otherwise, it returns NULL (i.e., 0) if it is not found. Write **`strchr(str, c)`** as a MIPS assembly function. Be sure to include the procedure prolog and epilog code. (10 marks)

ANSWER:

```
strchr:
        move    $v0, $a0
LOOP:
        lw      $t0, 0($v0)
        beq     $t0, $zero, RET0
        beq     $t0, $a1, RETv
        addi    $v0, $v0, 1
        j       LOOP
RET0:
        li      $v0, 0
RETv:
        jr      $ra
```

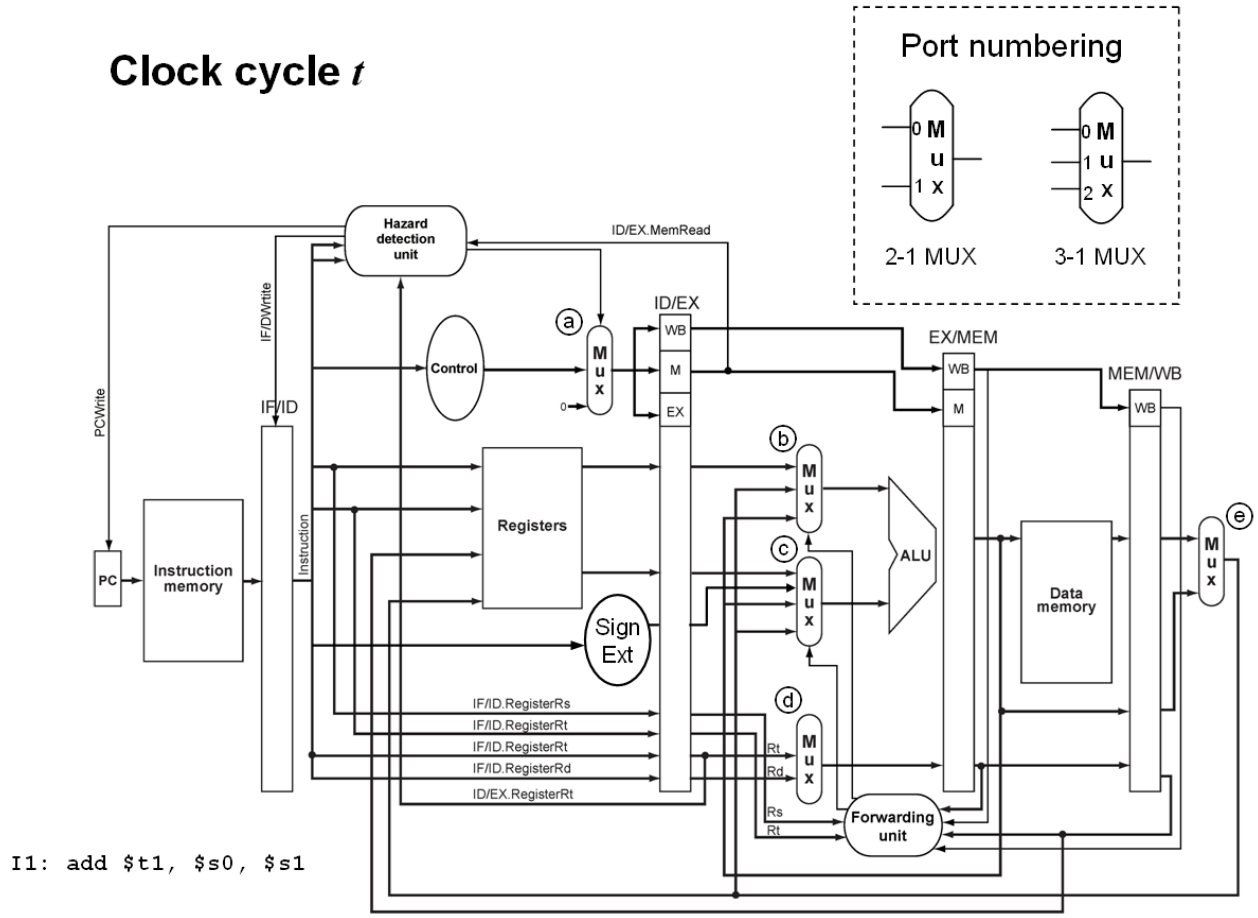
(3b) Write a MIPS assembly function **`mul(a, b)`** will multiply take two 16-bit *unsigned* integers, **`a`** and **`b`** and returns a 32-bit integer that is **`a`**×**`b`**. Inside the routine, you are *not* to use the multiply instruction, i.e., implement the standard multiplication algorithm that loops through the bits of the multiplier, summing up the (shifted) multiplicand. (10 marks)

ANSWER:

```
mul:
      move   $v0, 0
      move   $t0, 0
LOOP:
      andi   $t1, $a1, 1
      beq    $t1, $zero, CONT
      add    $v0, $v0, $a0 # partial product
CONT:
      srl    $a1, $a1, 1
      sll    $a0, $a0, 1   # multiplicand * 2
      addi   $t0, $t0, 1
      slti   $t1, $t0, 16
      bne    $t1, $zero, LOOP
      jr     $ra
```

QUESTION 4 (20 marks)

Consider the following pipelined data path at clock cycle *t*. (The multiplexor's inputs are labeled as per the diagrams in the box with the dotted border.) The instruction **`I1`**, namely **`add $1, $s0, $s1`**, is currently at the instruction fetch stage.



After **`I1`**, the following instructions enter the pipeline in order:

```
I2: lw $t2, -4($t1)
I3: sub $t3, $t2, $s5
I4: beq $t3, $t6, A
```

`I2` starts execution in the IF stage in cycle *t*+1, and **`I3`** starts at cycle *t*+2, and so on. Answer the following questions:

(4a) For the following situation, what are the values of the respective multiplexor's control input? (In other words, which input port of the respective multiplexor will be selected to become the output?) Assume that the instructions executing/executed prior to cycle *t* has no effect over the current execution (other than the default PC = PC+4). (8 marks)

ANSWER:

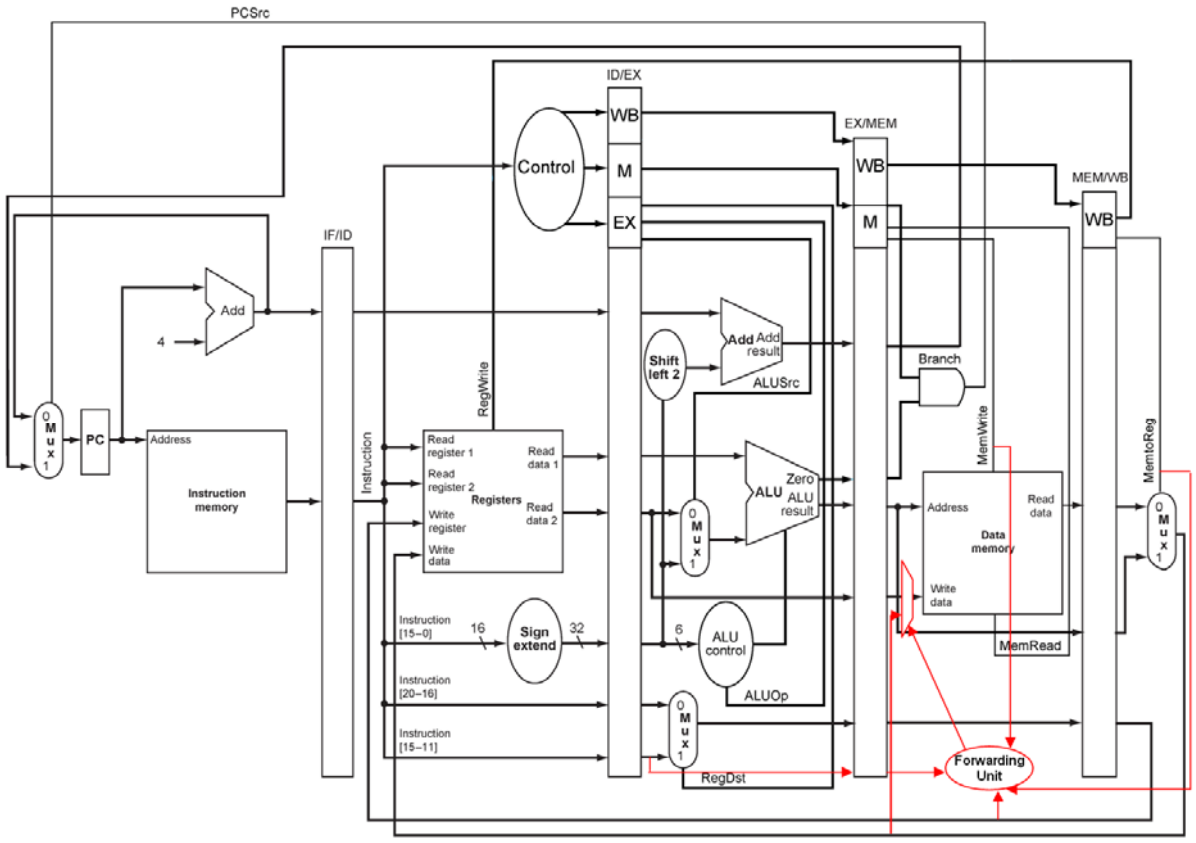
Cycle	MUX	Port	Reason
t+1	ⓐ	0	Normal control.
t+2	ⓐ	0	Normal control
	ⓑ	00	Rs to ALU port 1
	ⓒ	00	Rt to ALU port 2
	ⓓ	1	Rd
t+3	ⓐ	1	Inject NO-OP
	ⓑ	00	Rs to ALU port 1
	ⓒ	01	Sign extended immediate to ALU port 2
	ⓓ	0	Rt
t+4	ⓐ	0	Normal control
	ⓑ	00	NO-OP – doesn't matter
	ⓒ	00	NO-OP
	ⓓ	0	NO-OP
	ⓔ	1	ALU result to register
t+5	ⓐ	0	Normal control
	ⓑ	10	ALU port 1 takes input from WB
	ⓒ	00	Rt
	ⓓ	1	Rd
	ⓔ	0	Memory read result to go to registers

(4b) It is possible that a load instruction loads into a register that is immediately followed by a store instruction that stores this register's content into a different address. For example:

```
lw   $t0, 4($s0)
sw   $t0, 16($s3)
```

What is needed is a forwarding mechanism that will resolve this hazard. In the diagram below, sketch the connections and circuitry needed for such kinds of forwarding. (Please use a blue or red pen so that your answer will be well contrasted against the black lines of the original drawing.) (5 marks)

ANSWER:



(4c) Give the conditions for detecting such a hazard as well as resolving it. (4 marks)

ANSWER:

If EX/MEM.MemWrite AND
EX/MEM.Rt = MEM/WB.Rt AND
MEM/WB.MemtoReg THEN
ForwardMEM = 10

(4d) What if the code involved is:

```
lw  $t0, 4($s0)
sw  $t4, 16($t0)
```

How is this data hazard resolved? (3 marks)

ANSWER:

This is resolved by the standard MEM to ALU forwarding.

QUESTION 5 (15 marks)
A machine with byte addresses and a word size of 32 bits and address width of 32 bits has a two-way set associate cache with 4 blocks, each consisting of 2 words.

(5a) Given the C program below, and assuming that array `A` starts at memory hexadecimal location `0x2000` while array `B` starts at memory hexadecimal location `0x4020`. Assume that the cache is initially empty. Fill in the content of the cache immediately after the first time the cache entries are all used up. Use the notation M[*i*] to denote the word at memory address *i*. (*i* may be in hexadecimal.) (6 marks)

```
for (i=0; i<100; i++){
  B[i]= A[i] + i;
}
```

ANSWER:

Index	Tag	Word 0	Word 1
0	0x100	M[0x2000]	M[0x2004]
	0x201	M[0x4020]	M[0x4024]
1	0x100	M[0x2008]	M[0x200c]
	0x201	M[0x4028]	M[0x402c]
2	0x100	M[0x2010]	M[0x2014]
	0x201	M[0x4030]	M[0x4034]
3	0x100	M[0x2018]	M[0x201c]
	0x201	M[0x4038]	M[0x403c]

(5b) After the loop is completely executed, how many misses and hits would there be to the cache? (4 marks)

ANSWER:

There are 200 accesses, 100 hits and 100 misses.

(5c) Modify the loop that so that it will perform very badly in this cache, i.e. has the maximum number of misses. Explain why. (5 marks)

ANSWER:

```
for (i=0; i<100; i += 2){
  B[i]= A[i] + i;
}
```

Every access will miss because no spatial or temporal locality.

QUESTION 6 (10 marks)
A machine has a physical address width of 32 bits and a virtual address width of 32 bits. A page is 8192 bytes.

(6a) What is the maximum number of pages that can be supported? (2 marks)

ANSWER:

Maximum number of pages = 2^{32} / 8192 = 524288

(6b) Given the following content of a full associative 4-entry TLB:

Valid Bit	Virtual Page Number	Physical Page Number	Dirty	Ref	Access
0	0x70581	0x10112	1	20	rw
1	0x624F5	0x856	0	1	rwx
1	0x40A6A	0x40AA	0	8	rw
1	0x40A62	0x5A62	1	9	rw

Suppose now there is an access to the memory location at virtual address 0x40A62E90, what is the physical address that will be accessed? (3 marks)

ANSWER:

0x40A62E90 = 010|0 000|0 101|0 011|0 001|0 1110 1001 0000

Page number = 0x20531 – not found, page fault.

(6c) Suppose now there is an instruction fetch from the memory location at virtual address `0x101120C0`, which is located at physical memory location `0x900C0`, what is the content of the TLB after the operating system serviced the page fault? (5 marks)

ANSWER:

Valid Bit	Virtual Page Number	Physical Page Number	Dirty	Ref	Access
1	0x8089	0x48	0	1	rw
1	0x624F5	0x856	0	1	rwx
1	0x40A6A	0x40AA	0	8	rw
1	0x40A62	0x5A62	1	9	rw

`0x101120C0 = 000|1 000|0 000|1 000|1 001|0 0000 1100 0000`
Virtual page number = `0x08089`

Physical page number is `0x900C0 = 100|1 000|0 0000 1100 0000`
i.e., `0x48`

=== END OF PAPER ===