Indian Institute of Information Technology Allahabad

Review Test - Third Component (C3) (May 2019)

Second semester B.Tech (IT)

Course Name	Course Code	Date of Exam	MM	Time
Computer organization and Architecture	ICOA230C	May 06, 2019	75	3 Hr

Important Instructions*: All questions are compulsory.*

1. **(5+5+5= 15 marks)**

   (a) Given the follow chunk of code, to be executed on a byte-addressable computer with a total memory of 1Mega-Bytes. It also features a 16 Kilo-Bytes Direct-Mapped cache with 1 Kilo-Bytes blocks. Assume that the cache is initially empty.

```
#define NUM_INTS 8192   // 2^13
int A[NUM_INTS];        // A lives at 0x10000
int i, total = 0;
for (i = 0; i < NUM_INTS; i += 128) {
A[i] = i;            // Line 1
}

for (i = 0; i < NUM_INTS; i += 128) {
total += A[i];       // Line 2
}
```

   i. How many bits make up a memory address on this computer?
   ii. What is the #Tag bits : #Index bits : # Offset bits breakdown?
   iii. Calculate the cache hit rate for the line marked Line 1
   iv. Calculate the cache hit rate for the line marked Line 2

Solution:

i. $log_2(1MB) = log_2(2^{20}) = 20$

ii. Offset $= log_2(1KiB = log_2(2^{10}) = 10$
Index $= log_2(16KiB/1KiB) = log2(16) = 4$
Tag = 20-4-10 = 6

iii. The integer accesses are 4x128 = 512 bytes apart, which means there are 2 accesses per block. The first accesses in each block is a compulsory cache miss, but the second is a hit because *A[i]*and *A[i+128]* are in the same cache block resulting in a hit rate of 50%.

iv. The size of A is 8192x4 = 215bytes. This is exactly twice the size of our cache. At the end of Line 1, we have the second half of A inside our cache, but Line 2 startswith the first half of A. Thus, we cannot reuse any of the cache data brought infrom Line 1 and must start from the beginning. Thus our hit rate is the same as Line 1 since we access memory in the same exact way as Line 1. We do not have to consider cache hits for total, as the compiler will most likely store it in a register.Resulting in a hit rate of 50%.

   (b) Consider a 2-way set associative cache and 8-bit address space. There are 8 Bytes blocks and cache size is of 32 Bytes. Classify each of the following accesses as a cache hit (H), cache miss (M) or cache miss with replacement(R). For any misses, list out which type of miss it is.

Address	Tag : index : Offset bits	Hit/Miss/Replace
0000 0100		
0000 0101		
1100 1000		

What is the hit rate of our above accesses?

Solution:

```
 For this solution, we assume that we have an LRU replacement policy.

In general,this is not necessarily the case.

0000 0100     Tag 0000, Index 0, Offset 100 - M, Compulsory

0000 0101     Tag 0000, Index 0, Offset 101 - H

1100 1000     Tag 1100, Index 1, Offset 000 - R, Capacity

Hit Rate : #hits / #access = 1/3
```

(c) Consider a two level cache system. For 100 memory references, 20 misses in the first level cache and 6 miss in the second level cache. Miss penalty from L2 cache to memory is 40 cycles. The hit time of L2 cache is 4 cycles and hit time of the L1 cache is 1 clock cycle. What is the average memory access time (in cycle)?

Solution:

```
 For a two level cache L1 and L2 :
 average memory access time = L1 hit rate * Hit time +
 L1 miss rate * l2 hit rate * ( L1 hit time + L2 hit time)
 + L1 miss rate * L2 miss rate *
 (L1 hit time + L2 hit time + main memory access time)

 = 80*1 + 14 (1+4) + 6 (1+4+40)
 = 420 cycles
```

2. **(3+2+5+5= 15 marks)**

(a) A processor has 40 distinct instructions and 24 general purpose registers. A 32-bit instruction word has an opcode, two registers operands and an immediate operand. Find the number of bits available for the immediate operand field.

Solution: No. of bits required for 40 distinct instructions = $log_2(40)$ = 6 bits
No. of bits required for 24 GPR = $log_2(24)$ = 5 bits
Given opcode = 6 bits
Therefore, the remaining number of bits = 32 - (6+5+5) = 16 bits is used for immediate operand

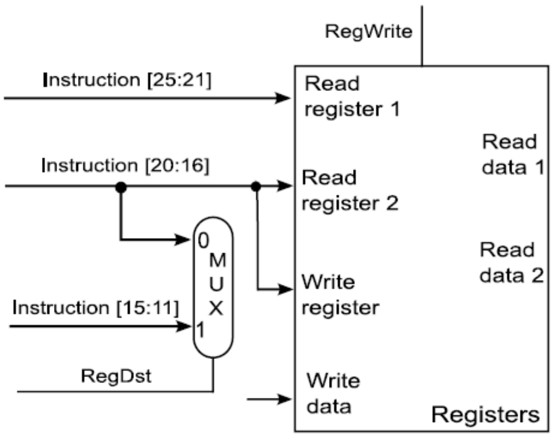
(b) Suppose a processor does not have any stack pointer register. Which of the following statements is true? Justify

i. It cannot have subroutine call instruction
ii. It can have subroutine call instruction, but no nested subroutine calls.
iii. Nested subroutine calls are possible, but interrupts are not.
iv. All sequences of subroutine calls and also interrupts are possible

Solution:

```
 It cannot have subroutine call instruction
```

(c) Suppose that a buggy implementation of the MIPS data path mis-wires the selection of the Write register number. Everything else in the data path operates normally.



i. Describe, in detail, how this error would affect the execution of the following instruction : add \$t0,\$t1, \$t2

ii. Describe, in detail, how this error would affect the execution of the following instruction : sw \$s0,0(\$s1)

Solution:

```
i. The sum of $t1 and $t2 is computed correctly,
but it is stored to $t2, not $t0

ii. Since sw doesnot write a value to any register,
the error has no effect whatsoever.
```

(d) Consider a RISC machine where each instruction is 4 bytes long.Conditional and unconditional branch instructions use PC-relative addressing mode with Offset specified in bytes to the target location of the branch instruction. Also, the Offset is always with respect to the address of the next instruction in the program sequence. Consider the following instruction sequence:
Instruction i: ADD R2,R3,R4
Instruction i+1: SUB R5,R6,R7
Instruction i+2: SEQ R1,R9,R10
Instruction i+3: BEQZ R1,Offset
If the target of the branch instruction is *i*, what is the the decimal value of *Offset*

Solution:

```
Assume that instruction "i" starts from memory address X.
Address of instruction i+1 = X + 4
Address of instruction i+2 = X + 8
Address of instruction i+3 = X + 12
Address of instruction i+4 = X + 16
So, Offset = X - (X + 16) = -16
```

3. **(5+5+5 = 15 marks)**

(a) Suppose you have to execute the following arithmetic operations in MIPS X = (P+Q)/(R-S) Use 3- address, 2-address, 1-address, and Zero-address instruction modes to solve the above problem.

Solution:

```
3 address instructions :
----------------------

add R1,P,Q
sub R2,R1,S
div X,R1,R2


2 address instructions :
----------------------

mov R1,P
add R1,Q
mov R2,R
sub R2,S
div R1,R2
mov X,R1

1 address instructions :
----------------------

load R1
sub S
store R2
load P
add P
div T
store X

0 address instructions :
----------------------
push R
push S
sub
push P
push Q
add
div
pop X
```

(b) Pseudo-instructions give MIPS a richer set of assembly language instructions than those supported directly by the hardware. Explain how each of the following pseudo-instruction can be implemented using only the MIPS32 instructions:

i. *sge $rd, $rs, $rt*
Operation expected of the instruction : GPR[rd] =(GPR[rs] >= GPR[rt]? 1 : 0)

ii. *bge $rs, $rt label*
Operation expected of the instruction : Branch to label if GPR[rs] >= GPR[rt]
(Hint for bge: try using slt instruction)

The MIPS 32 real instructions to be used is listed below (those are sufficient). If you need to use any"extra" registers, you may use the t-registers, $t0 , $t1, and so forth.

```
add   rd, rs, rt # GPR[rd] <--GPR[rs] + GPR[rt]
addi  rd, rs, imm16 # GPR[rd] <--GPR[rs] + imm16 (sign extended)
andrd, rs, rt # GPR[rd] <--GPR[rs] AND GPR[rt](bitwise)
andi  rd, rs, imm16 # GPR[rd] <--GPR[rs]OR GPR[rt]  (bitwise)
beq   rs, rt, label # if GPR[rs] == GPR[rt], jump to label

bne   rs, rt, label # if GPR[rs] != GPR[rt], jump to label

orrd, rs, rt # GPR[rd] <--GPR[rs] OR GPR[rt](bitwise)

ori   rd, rs, imm16 # GPR[rd] <--GPR[rs] OR imm16   (bitwise, sign extended)

subrd, rs, rt # GPR[rd] <--GPR[rs] -GPR[rt]

sllrd, rt, sa # GPR[rd] <--GPR[rs] << sa(logical shift)

slt   rd, rs, rt # GPR[rd] <--(GPR[rs] <GPR[rt] ? 1 : 0)
```

Solution:

```
 i.  BEFORE
    ------

    sge $rd, $rs, $rt

    AFTER
    -----
Here slt is used to set $rd to the wrong value,
and then flips the low bit to correct it:

slt   $rd, $rs, $rt # $rd = $rs < $rt ? 1 : 0
ori   $t0, $zero, 1 # $t0 = 1
subrd  $rd, $t0, $rd    # $rd = 1 -$rd


ii.  BEFORE
      ------
  bge $r1, $r2, LABEL   # branch if $r1 >= $r2

      AFTER
      -----
  slt $r3, $r1, $r2     # if $r1 >= $r2, then $r3 will be 0 (false)
  beq $r3, $r0, LABEL   # branch if previous condition was false

Notice this rewriting requires an additional register.
```

(c) In a MIPS program, assume that the program counter PC currently contains address *000000F0* (address is in hexadecimal) and program needs to jump to the label *LL*, which is labeling address *3FFFFFF0* (address is in hexadecimal). If you use an instruction like *j LL*, then you can easily do it, and this is fine when you are writing a program. However, in this case, it cannot be a basic machine instruction in the J-type instruction format. What may be the reason? [Hint: See how far the program counter has to jump]

Solution:

```
 jump address = (PC+4)[31:28], value_for_address_field, 00

So if the j LL instruction is at 0x000000F0, then it can only jump
to addresses that look like 0x0XXXXXXX
```

4. **(4+4+2 = 10 marks)**

(a) Consider an instruction pipeline with four stages with the stage delays 5 nsec, 6 nsec, 11 nsec, and 8 nsec respectively. The delay of an inter-stage register stage of the pipeline is 1 nsec. What is the approximate speedup of the pipeline in the steady state under ideal conditions as compared to the corresponding non-pipelined implementation?

Solution:

```
 Time elapsed for the non-pipelined architecture = 5+6+11+8 = 30ns

 Time elapsed for the pipelined architecture = 11+1
  = 12 ns

  Speed up  = 30/12 = 2.5
```

(b) The instruction pipeline of a RISC processor has the following stages: Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Perform Operation (PO) and Write back(WB), The IF, ID, OF and WB stages take 1 clock cycle each for every instruction. Consider a sequence of 100 instructions. In the PO stage, 40 instructions take 3 clock cycles each, 35 instructions take 2 clock cycles each, and the remaining 25 instructions take 1 clock cycle each. Assume that there are no data hazards and no control hazards. Find the number of clock cycles required for completion of execution of the sequence of instruction.

Solution:

```
Total no. of instructions = 100
No. of stages = 5
Total cycles required =
Total no. of cycles required for general case  + extra cycles required

= n+k-1 + extra cycles

= (100+5-1) + 40*(3-1) + 35*(2-1)
= 104 +115
219 cycles
```

(c) For a pipelined CPU with a single ALU, consider the following situations :
1) The *j+1*th instruction uses the result of *j*th instruction as an operand
2) The execution of a conditional jump instruction
3) The *j*th and *j+1*th instructions require the ALU at the same time.
Which of the above can cause a hazard? Justify

i. 1 and 2 only
ii. 2 and 3 only
iii. 3 only
iv. All the above.

Solution:

```
All the above
```

5. **(3+3+1+3 = 10 marks)**

(a) The two numbers given below are multiplied using the Booths algorithm.
Multiplicand : 0101 1010 1110 1110
Multiplier: 0111 0111 1011 1101
How many additions/Subtractions are required for the multiplication of the above two numbers?

Solution:

```
Booth's algorithm: first take 2's complement of given number
if number is negative, then append 0 into LSB.

Then, for each pair from LSB to MSB (add 1 bit at a time):
00 = 0, 01 = +1, 10 = -1, 11 = 0

Booth's algorithm is based on Multiplier which is already given in binary
representation: 0111 0111 1011 1101

= Now, append 0 into LSB of (0111 0111 1011 1101) =
0111 0111 1011 1101 0

Now Booth's code (add 1 bit at a time, from LSB to MSB):
= 01, 11, 11, 10, 01, 11, 11, 11, 10, 01, 11, 11, 11, 10, 01, 10
= +1 0 0 -1 +1 0 0 0 -1 +1 0 0 0 -1 +1 -1

Therefore, 4 subtractions and 4 additions,
total 8 additions/Subtractions are required
```

(b) Using two's complement division algorithm obtain the result for -7/3

Solution:

```
As discussed in class consider three regsiters A,Q and M
A = 1111   Q = 1001 and M=0011  : Intial Values

A Q

1110 0010  Shift
0010       Add
1111 0010  Restore

1110 0100  Shift
0001       add
1110 0100  Restore

1100 1000  Shift
1111       add
1111 1001  Set Q0 = 1

1111 0010  Shift
0010       add
1111 0010  Restore

Quotient (Q) = -2
remainder (A) = -1
```

(c) Which of the following statements are false for floating-point multiplication and division?

i. Alignment of the mantissa values is required
ii. Normalization of the result after multiplication or division is required
iii. After adding or subtracting the exponents, a corrective subtraction or addition step is required.
iv. None of the above

Solution:

```
Correct answer is (i).
Mantissa alignment is required only for addition and subtraction.
(ii) and (iii) are true for multiplication and division.
```

(d) Compute the following floating-point sum:1313.3125 + 0.1015625

Solution:

```
1313.3125 = 1.01001000010101 * 2^10
0.1015625 = 1.101 * 2^-4

normalize to same exponent:

101001000010101.0 * 2^-4
      1.1010  * 2^-4
-----------------------------
101001000010110.1010  * 2^-4

renormalized: 1.010010000101101010 *2^10
new mantissa: 010010000101101010
exponent: 10 + 127 = 137 = 10001001

Ans: 0 10001001 01001000010110101000000 = 0x44A42D40
```

6. **(3+2+3+2 = 10 marks)**

   (a) Data is arriving at 4000 characters per second. If the device is served by polling, and each character must be read before another one is received, what is the maximum time allowed between polls? If each poll requires 10 microseconds to complete, what fraction of the CPU time is always being used up when the serial port is idle?

   Solution:

   ```
   Data arrival rate of each character = 1sec/4000 = 0.00025 sec = 250 microsec
   This is the maximum time allowed between two polls.

   Serial port remains idle for 250 micro sec (inter arrival time of characters).

   Polling requires 10 microseconds and it is performed
   within the arrival time of two successive characters.

   Therefore, fraction of time polling is performed  =
   10microseconds/250 microseconds * 100 which is 4% of
   the idle time spent on polling
   ```

   (b) As can be seen in the question 6.(a) polling has a negative impact on processor time. A better solution would be to use interrupts. What is an interrupt? What happens during execution of an interrupt?

Solution:

```
An interrupt is the automatic transfer of software execution
in response to a hardware event that is asynchronous with the
current software execution. This hardware event is called a
trigger. The hardware event can be due to an external I/O device
(like the UART input/output) or an internal event (like
bus fault, memory fault, or a periodic timer). When the hardware
needs service, it will request an interrupt by setting its
trigger flag. The processor stops its current execution and
program control is transferred to a sub-routine ( interrupt
service routine (ISR)) by generating CALL signal and after
executing sub-routine by generating RET signal again program
control is transferred to main program from where it had stopped.
When microprocessor receives interrupt signals, it sends an
acknowledgement (INTA) to the peripheral which is requesting
for its service
```

(c) Data is being read from an interface that reads 100,000 characters per second. The overhead to service an interrupt and return control to the interrupted program is 20 microseconds. Can this device use an ISR to transfer each character?

Solution:

```
No, the device cannot used as an ISR to transfer each character.
Given, the data rate = 100,000 characters per second
Therefore, time taken for each character = 1/100000 = 0.000001 seconds
= 10 microseconds. Since the interrupt overhead is 20 microseconds
which overshoots the inter character transfer time.
```

(d) You are designing a simple furnace controller. It uses a processor to read the temperature sensor and turn *ON* and *OFF* the heater in response to the temperature. What is better strategy for data reading - polling or use of interrupt? Justify.

Solution:

```
Interrupt would be a better choice for data reading since
the host would need to query the Status register if and only if
new data has to be read. Depending on the speed of the host
processor, this can result in much higher efficiency,
lower overall power usage, or both
```