**C and MIPS Assembly Programming**

Following is the definition of the Ackermann function (thanks, Wikipedia), which is defined over the domain of non-negative integers:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

a) Complete the following C function, `ack`, so that it computes the Ackermann function correctly.

```
unsigned int ack(unsigned int m, unsigned int n)
{
  unsigned int answer;

  if(m == 0)
    answer = n+1;
  else if(n == 0)
    answer = ack(m-1, 1);
  else
    answer = ack(m-1, ack(m, n-1));

  return answer; }
```

b) Write MIPS assembly that implements ack correctly.  m and n are passed in `a0` and `a1`, respectively.  We've provided the prologue and epilogue for you.  `s0` and `s1` are saved and restored, so you can safely use them if you need to preserve values across function calls.

```
addiu $sp, $sp, -12
sw    $ra, 0($sp)
sw    $s0, 4($sp)
sw    $s1, 8($sp)

# the m == 0 case
bne   $a0, $0, elseif
addiu $v0, $a1, 1  # answer = n + 1
j     epilogue

# the m != 0, n == 0 case
elseif:
bne   $a1, $0, else
addiu $a0, $a0, -1  # arg0 = m-1
li    $a1, 1        # arg1 = 1
jal   ack           # answer = ack(m-1, 1)
j     epilogue

# the final case
else:
addiu $a1, $a1, -1  # arg1 = n-1.  arg0 is already m.
move  $s0, $a0      # preserve m
jal   ack           # v0 = ack(m, n-1)
addiu $a0, $s0, -1  # arg0 = m-1
move  $a1, $v0      # arg1 = ack(m, n-1)
jal   ack           # answer = ack(m-1, v0)

epilogue:
lw    $ra, 0($sp)
lw    $s0, 4($sp)
lw    $s1, 8($sp)
addiu $sp, $sp, 12
jr    $ra
```

**AMAT**

The average memory access time (AMAT) is:

AMAT = Hit $\text{Time}_{L1}$ + Miss $\text{Rate}_{L1}$ x Miss $\text{Penalty}_{L1}$

(a) Write down the AMAT equation for a two level cache (Use HT for hit time, MR for Local Miss Rate, and MP for Miss Penalty):

AMAT = $HT_{L1}$ + $MR_{L1}$ x ($HT_{L2}$ + $MR_{L2}$ x $MP_{L2}$)

(b) Given the following specifications:
For every 1000 CPU-to-memory references:
40 will miss in L1\$
20 will miss in L2\$
L1\$ hits in 1 clock cycle
L2\$ hits in 10 clock cycles
Main memory access is 100 clock cycles
There are 1.3 memory references per instruction
Ideal CPI is 1.

Answer the following questions:
(i) What is the local miss rate in the L2\$?

20 / 40 = 50%

(ii) What is the global miss rate in the L2\$?

20 / 1000 = 2%

(iii) What is the AMAT with both levels of cache?

1 + 4% x (10 + 50% x 100) = 1 + 4% x 60 = 1 + 2.4 = 3.4

(iv) What is the AMAT for a one-level cache without L2\$?

1 + 4% x 100 = 5

(v) What is the average memory stalls per reference in the system of question (iii)?

$MR_{L1}$ x ($HT_{L2}$ + $MR_{L2}$ x 100) = 2.4

(vi) What is the average memory stalls per reference in the system of question (iv)?

$MR_{L1} \times 100 = 4$

(vii) What is the average memory stalls per instruction in the system of question (iii)?

average memory stalls per $\text{reference}_{(iii)}$ x 1.3 memory references per instruction
= 2.4 x 1.3 = 3.12

(viii) What is the average memory stalls per instruction in the system of question (iv)?

average memory stalls per $\text{reference}_{(iv)}$ x 1.3 memory references per instruction
= 4 x 1.3 = 5.2

(ix) What is the performance of the system of question (iv) verses that of question (iii)?

$\text{runtime}_{(iv)} / \text{runtime}_{(iii)}$

$$= \frac{CPI_{(iv)} \times \text{Cycle Time} \times \text{number of instructions}}{CPI_{(iii)} \times \text{Cycle Time} \times \text{number of instructions}}$$

$$= \frac{CPI_{(iv)}}{CPI_{(iii)}} = \frac{CPI_{ideal} + \text{average memory stalls per instruction}_{(iv)}}{CPI_{ideal} + \text{average memory stalls per instruction}_{(iii)}}$$

= (1 + 5.2) / (1 + 3.12) = 1.5x

**MIPS 5 stage pipeline**

Consider a variation on the canonical MIPS 5 stage pipeline, which doesn't have any bypass paths, doesn't use branch delay slots, and resolves branches in the execute stage.

For the following code sequence, answer questions (a) - (c).

addu $t0, $a0, $a1
sllv $t1, $t0, $a2

(a) How many cycles does the processor stall?

2 cycles

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D | X | M | W | | | | | | | |
| | F | D | X | M | W | | | | | | |
| | | F | D | X | M | W | | | | | |
| | | | F | D | X | M | W | | | | |

(b) Assume we have added a forwarding path from the beginning of the writeback stage to the beginning of the execute stage. How many cycles does the processor stall?

1 cycle

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D | X | M | W | | | | | | | |
| | F | D | X | M | W | | | | | | |
| | | F | D | X | M | W | | | | | |

(c) Assume we have implemented all forwarding paths. How many cycles does the processor stall?

0 cycles

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D | X | M | W | | | | | | | |
| | F | D | X | M | W | | | | | | |

For the following code sequence, answer questions (d)-(h).

```
addu $t3, $t4, $t5
beq $t0, $t1, label
srlv $s0, $s1, $s2
addu $t4, $t4, $t4
```

(d) When $t0 != $t1, how many instruction fetches are wasted in the original processor?

0

(e) When $t0 == $t1, how many instruction fetches are wasted in the original processor?

2

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D | X | M | W | | | | | | | |
| | F | D | X | M | W | | | | | | |
| | | F | D | X | M | W | | | | | |
| | | | F | D | X | M | W | | | | |

(f) Assume the branches are resolved in the decode stage. When $t0 == $t1, how many instruction fetches are wasted?

1

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D | X | M | W | | | | | | | |
| | F | D | X | M | W | | | | | | |
| | | F | D | X | M | W | | | | | |

(g) Assume the processor implements branch delay slots, and branches are resolved in the decode stage. Reorganize the code sequence to minimize number of wasted instruction fetches when $t0 == $t1.

```
beq $t0, $t1, label
addu $t3, $t4, $t5
srlv $s0, $s1, $s2
addu $t4, $t4, $t4
```

(h) Given the code sequence in (g), how many instruction fetches are wasted when $t0 == $t1?

0

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D | X | M | W | | | | | | | |
| | F | D | X | M | W | | | | | | |
| | | F | D | X | M | W | | | | | |

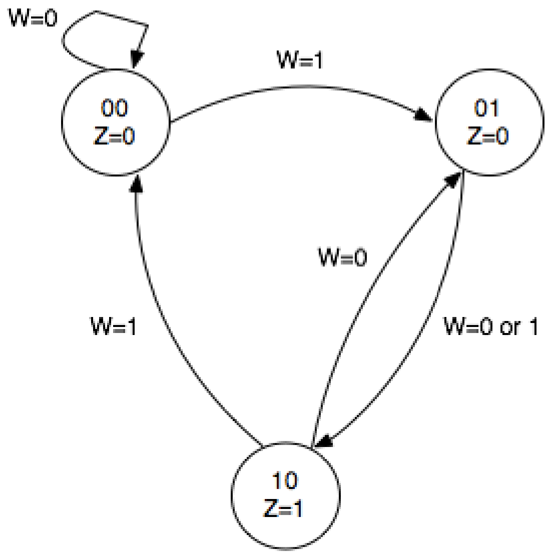
(i) Given all the optimizations above (full forwarding paths, resolve branches in the decode stage, and branch delay slots), is there a situation where the processor needs to stall? If so, provide an example sequence of instructions that will interlock.

```
lw $t0, 40($s0)
addu $t1, $t0, $a0
```

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| F | D | X | M | W | | | | | | | |
| | F | D | X | M | W | | | | | | |
| | | F | D | X | M | W | | | | | |

**FSM**

Given the following state machine, answer questions (a)-(d).



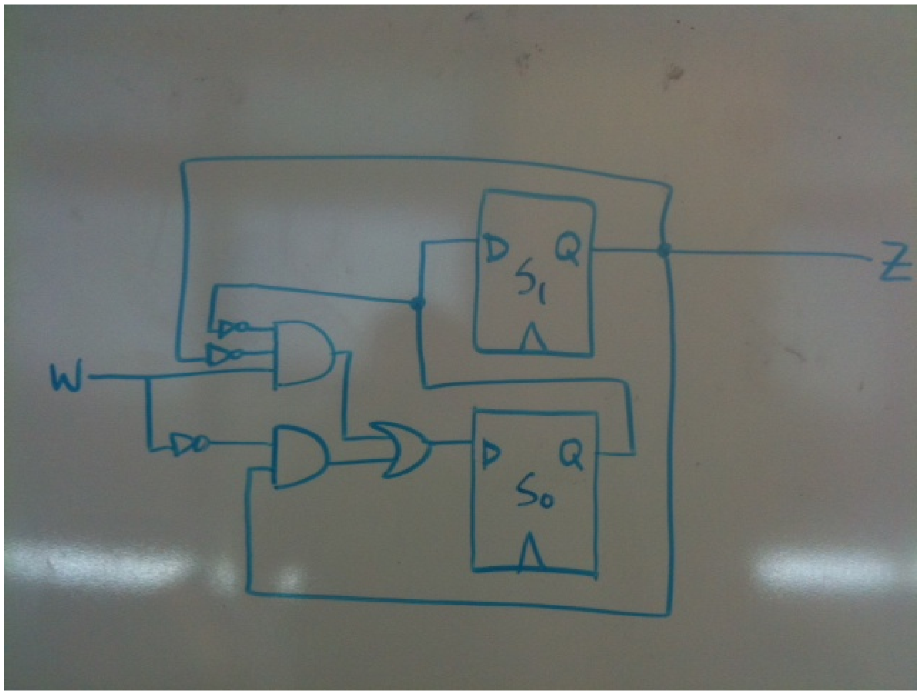
(a) Fill in the following truth table:

| S1 | S0 | W | NS1 | NS0 | Z |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | X | X | X |
| 1 | 1 | 1 | X | X | X |

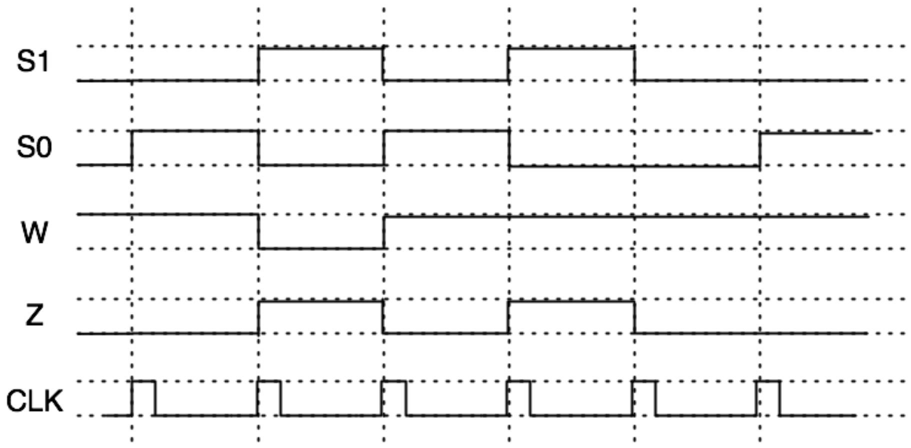
(b) Write boolean expressions that implement the NS1, NS0, and Z function.

$$NS_1 = S_0$$
$$NS_0 = S_1\overline{W} + \overline{S_1}\,\overline{S_0}\,W$$
$$Z = S_1$$

(c) Implement the finite-state machine by finishing the diagram.



(d) Draw the waveform.



**Virtual Memory**
Consider a 32-bit MIPS machine like the one we studied in class with 4 GB of physical main memory. The machine's virtual memory system uses 4 KB pages. PTEs (page table entries) have dirty, valid, readable, writable, & executable bits along with the PPN (physical page number). The machine uses a flat page table (with all entries present) to perform translation in the event of a TLB (translation lookaside buffer) miss.

(a) How big is a PTE in bits? Assume that the size is rounded up to the next byte and the extra bits are reserved for use by the OS. Name the bits in the PTE.

PPN = 20 bits
dirty = 1 bit
valid = 1 bit
readable = 1 bit
writable = 1 bit
executable = 1 bit

total = 25 bits
round up to next byte => 32 bits, so we also have 7 bits reserved for use by the OS.

(b) How large is a page table in this system?

4 GB virtual address space / 4 KB pages = 1 M pages
1 M pages * 4 bytes per page table entry = 4 MB

(c) Assume that a process must have, at minimum, a single data page and a single code page. Remember to account for the space required by the page tables. What is the maximum number of processes we can run at a time without paging to disk (ignoring the pages used by the operating system itself)?

minimum process size = 4 MB for the page table + 4 KB data + 4 KB code = (2^22+2^13) bytes
maximum # of processes = 4 GB physical address space / (2^22+2^13) bytes = 1022

**Set-Associative caches**

(a) Given a 2-way set-associative cache, LRU replacement policy, initially empty, and the following memory access pattern (all byte addresses and 32-bit word accesses, 32-bit addresses)

8 0 4 32 36 8 0 4 16 0

What is the hit rate, miss rate, and what blocks are in the cache after these accesses if the cache has 4 32-bit blocks?

*hit rate: ____20%_____  miss rate: ____80%_____*

*blocks at end (write the appropriate full byte addresses (NOT the tag) in the appropriate blocks, or "EMPTY" if the block is empty):*

| | way 0 | way 1 |
|---|---|---|
| set 0 | 0 | 16 |
| set 1 | 4 | 36 |

(b) Consider a write-allocate, write-through, 4-way set-associative cache with 16 byte blocks and $64*2^{10}$ bytes of data bits. Assume a byte-addressed machine with 32-bit addresses.

a. Partition the following address and label each field with its name and size in bits.

31 0

| |
|---|
| |

| | | |
|---|---|---|
| [31:14] tag | [13:4] index | [3:0] offset |

(i) Given the address $\text{DEADBEEF}_{\text{hex}}$, what is the value of the index, offset, and tag? (Write your answers in hexadecimal.)

*index = 0x___________3EE_____________*

*offset = 0x___________F_____________*

*tag = 0x___________37AB6_____________*

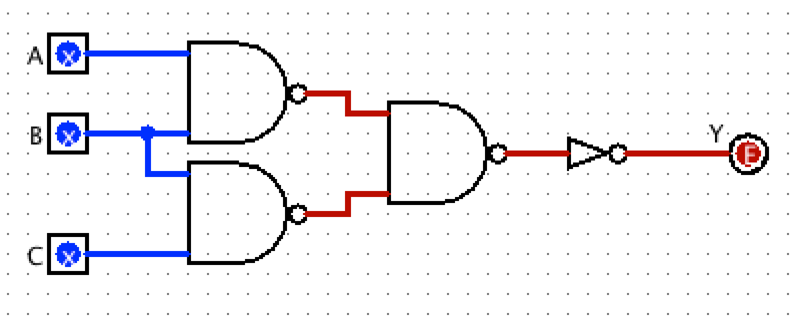
(ii) How many cache management bits are there for each block? List them.

tag =18 bits
valid = 1 bit
total = 19 bits / block

(iii) How many bits comprise the cache?
cache management bits = 19b x 4K blocks = 76 Kbits
data bits = 512 Kbits
total = 76 + 512 = 588 Kbits

**Digital Logic**

Consider the following digital circuit, which implements a combinational logic function:



a) Fill in the following truth table for the above circuit.

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

b) Write a sum-of-products Boolean expression that corresponds to this function.

$$Y = \overline{A \bullet B}\ \overline{B \bullet C}$$
$$Y = (\overline{A} + \overline{B}) \bullet (\overline{B} + \overline{C})$$
$$Y = \overline{A}\ \overline{B} + \overline{A}\ \overline{C} + \overline{B} + \overline{B}\ \overline{C}$$

(An answer that derived the canonical, 5-term expression from the truth table would also be correct, of course.)

c) Write the most simplified sum-of-products Boolean expression that implements this function.

$$Y = \overline{B} + \overline{A}\ \overline{C}$$