EECS 370 Exam 1
Fall 2002

Name: ____________________________________ unique name: _____________

Sign the honor code:

I have neither given nor received aid on this exam nor observed anyone else doing so.

____

Scores:

#	Points
1	/40
2	/15
3	/45
Total	**/100**

NOTES:

- Open book and Open notes
- Calculators are allowed, but no PDAs, Portables, Cell phones, etc.
- Don't spend too much time on any one problem.
- You have about 120 minutes for the exam.

Section I: Short Answer

1. What will be the result of execution of this program in LC2? Explain your answer. Be sure to indicate the value of any register that changes. (hint: 21626880 is 14A0000 in hex). **[5 points]**

```
start     .fill 21626880
          halt
```

2. Consider the following C structure.

```
struct LinkedList {
   int element;
   struct LinkedList *next;
  };
```

The following LC2K2 code is supposed to sum the value of all of the elements in the linked list. The pointer of the last element of the linked list points to location zero, and no other pointer points to that location. Fill in the missing gaps with the correct values to cause this program to work. **[10 points]**

```
        lw  0    3    start  Load pointer to 1st element.
        add 0    0    4
addLL   beq ___  0    done
        lw  ___  2    0
        add 2    4    4
        lw  ___  3    _______
        beq 0    0    addLL
done    halt
start   .fill ________
first   .fill 4          First element of the link list
        .fill second     pointer to the next element
second  .fill 10
............             (More of the data structure follows)
```

3. Given the initial values located in memory locations 100-103, what are the final contents of the memory after the following MIPS instruction sequence is executed? **[5 points]**

```
lb $1, 100($0)
lhu $2, 102($0)
sw $2 100($0)
sh $1 102($0)
```

Initial Memory Contents

Address	Value
100	0xA0
101	0xAA
102	0xF0
103	0x0F

Final Memory Contents

Address	Value
100	____
101	____
102	____
103	____

4. Translate the following C program into MIPS assembly. Let $r1 correspond to `a` and let $r2 correspond to `b`. If your program is longer than 6 instructions you will get zero points. **[5 points]**

```
if(a>b)
      a=5;
else
      b++;
```

5. Given below is a partial-MIPS function. Unfortunately, the author completely forgot to save and restore registers. Recall, that according to the MIPS calling convention, $si registers are designated as calle-save and $ti registers are caller-save (See page A-23 for additional details). At each label in the program (`start_fn, before_jal, after_jal, end_fn`) specify the registers that MUST be saved and/or restored at that point in order to satisfy the MIPS calling convention. *You should not save any registers that do not need saving.* You may specify your answers as "save $s0" or "restore $s0", you do not need to worry about allocating space on the stack to handle saving/restoring. "bar" is a function which follows the MIPS calling convention, but you don't know anything else about it. **[5 points]**

```
start_fn:

  lw $s0, 0($a0)
  lw $s1, 4($a0)
  lw $t0, 8($a0)
  lw $t1, 12($a0)
  add $s2, $s0, $s1
  add $t2, $t0, $t1

before_jal:

  jal bar

after_jal:

  sw $s2, 0($a0)
  sw $t2, 4($a0)

end_fn:

  jr $ra
```

6. Prove that the NOR gate is universal by showing how to build the AND, OR, and NOT functions using one or more two-input NOR devices. **[5 points]**

7. Given are the following two C files each containing one function and one global variable definition. What symbols does the symbol table for ***each file*** contain when the corresponding object file is constructed by an assembler? Note that an `extern` declaration specifies that the storage for the declared objects is defined elsewhere. It is similar to a function prototype but for variables instead. **[5 points]**

file1.c

```
int bar(int);
extern char c[];

int a;

int foo(int x)
{
    int b;

    reference to a
    reference to c
    call bar
    reference to b
}
```

File 1 symbol table

file2.c

```
extern double d[];

char c[100];

int bar (int y)
{
    char *e[100];

    reference to y
    reference to d
    reference to e
    reference to c
}
```

File 2 symbol table

Section II: MIPS single-cycle datapath

1. Consider the figure on page 358 of our text (also attached) and answer the following questions:

   - In front of one of the adders is a “shift left 2” box. What is the purpose of this box? Why doesn’t the LC datapath have one? **[3 points]**

   - Now consider computing the cycle time for that single-cycle processor. Using the following table, figure out the minimum cycle time required for each of the instructions listed. You may not overlap instructions in any way. You must indicate how you arrived at your answer. **[12 points]**

Action	Latency
Read from memory	10ns
Write to memory	5ns
Read from register file	2ns
Write to register file	1ns
ALU operation (any)	4ns
Adder	3ns
Write to PC	1ns
All other actions	0ns

Instruction	Actions taken which contribute to cycle time for the instruction	Minimum cycle time required for the instruction
BEQ		
ADD		
LW		
SW		

Section III: Design

Consider the 8-bit stack-based ISA named "HOB".

There are 3 instruction formats (bit 0 is the least-significant bit).

R-type instructions (add, nand):
  bits 7-5: opcode
  bits 4-0: unused (should all be 0)

I-type instructions (pushi):
  bits 7-5: opcode
  bits 4-0: value (a 5-bit, 2's complement number with a range of –16 to 15)

A-type instructions (push, pop, beq)
  bits 7-5: opcode
  bits 4-0: address (in the range of 0 to 31)

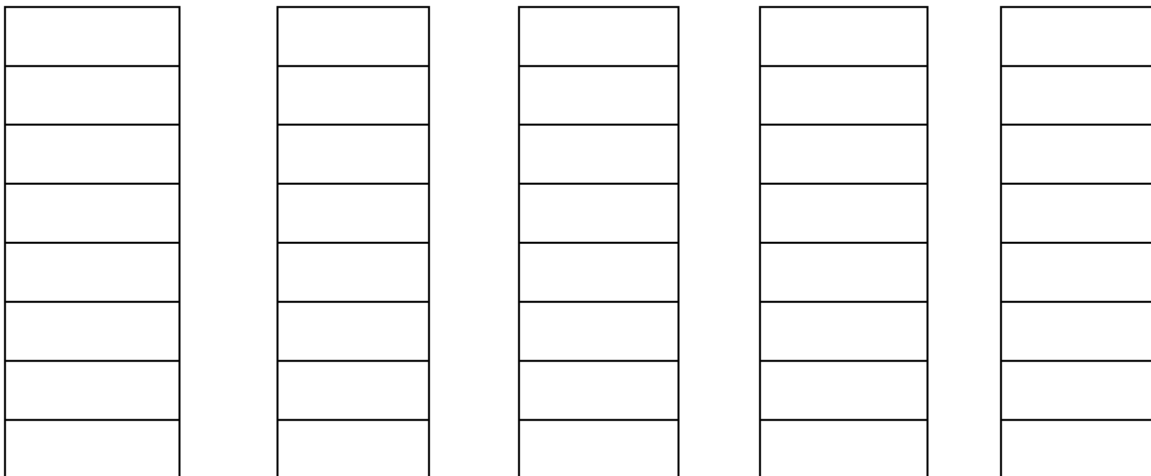
Assembly language Instruction	**Opcode in binary (bits 7, 6, 5)**	**Action**
add (R-type)	000	Pop the top two values off of the stack and push on the sum of those two values back on the stack.
nand (R-type)	001	Pop the top two values off of the stack and push on the bit-wise NAND of those two values back on the stack.
push (A-type)	010	Take the value stored at memory location specified by the address field and push it onto the stack
pop (A-type)	011	Take the value at the top of the stack and pop it. Place that value into memory location specified by the address field.
pushi (I-type)	100	Push the sign-extended value stored in the value field.
beq (A-type)	101	If the two values on the top of the stack are equal, pop them and branch to the location specified by the address field. Otherwise do nothing (don't change anything on the stack.)
halt (R-type)	110	Increment the PC (as with all instructions), then halt the machine (let the simulator notice that the machine halted).

1. Translate the following program into *hexadecimal*. **[6 points]**

```
pushi 0
push 5
beq 5
add
halt
```

2. Show value(s) stored in the stack *after* each time the following program encounters a beq *as well as* when the program halts. Assume the PC starts as zero initially. We've provided space for your answer, but be sure to label each stack. Draw your stack so the "top-of-stack" is on top. **[7 points]**

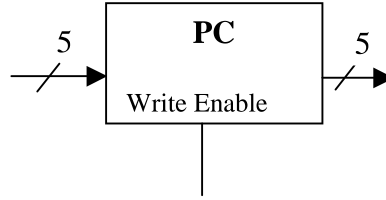
```
pushi 10
pushi 0
pushi 2
pushi -1
add
beq 12
pop 30
pop 29
pushi –2
push 29
push 30
pushi 0
pushi 0
beq 3
halt
```

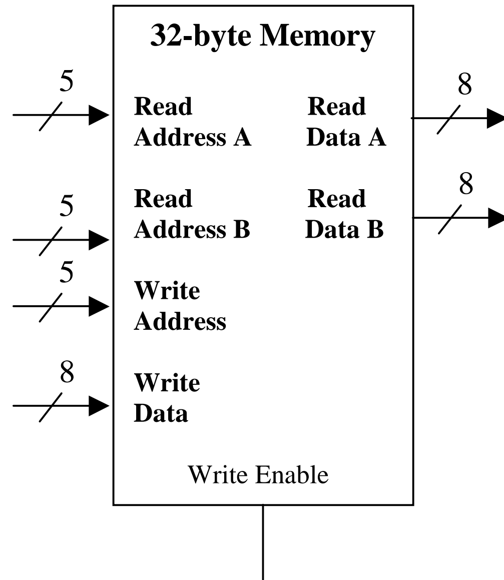


Bottom of stack

3. Say you have the following devices:

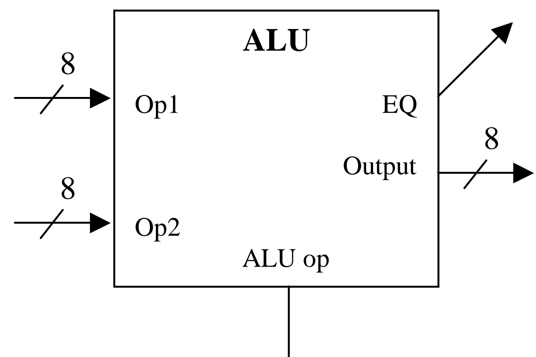
- **PC**: This is a 5-bit register with a write-enable control. When first powered up it initializes itself to zero. When the write-enable is 1, the device will store the input value on the next rising edge of the clock. (That is, at the end of the clock cycle.)





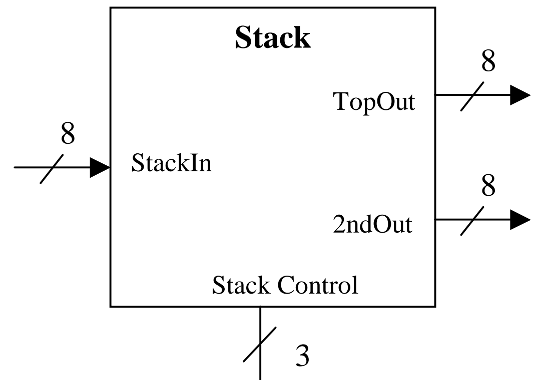
- **32-byte memory**: When an address is presented to the Read Address A input, the data stored in that address is immediately written to Read Data A. When an address is presented to the Read Address B input, the data stored in that address is immediately written to Read Data B. When Write Enable is a 1, Write Data is written to the address specified by Write Address on the rising edge of the system clock. (That is, at the end of the clock cycle)

- **ALU**: When ALUop is 1 the value Op1 + Op2 is placed on Output. When ALUop is 0 the bitwise NAND of Op1 and Op2 is placed on Output. Also, if Op1 and Op2 are equal then EQ=1, else EQ=0.



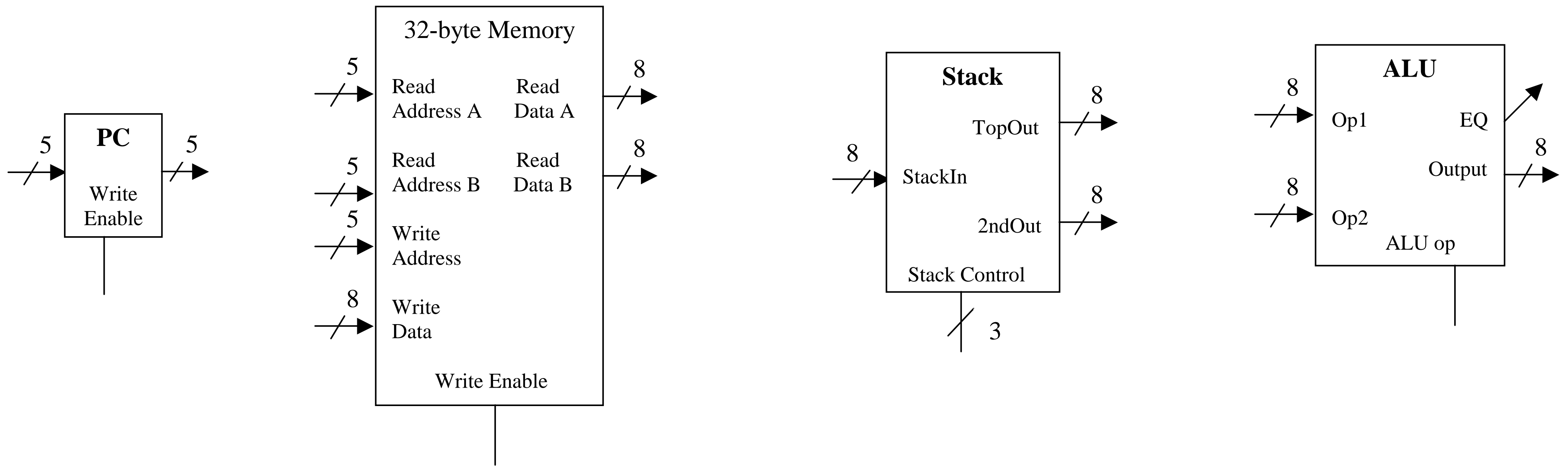
- **Stack:** The top value of the stack is placed on TopOut, the second highest value is placed on 2ndOut. Further the control signal has the following options:
  - 0: do nothing
  - 1: pop the top value
  - 2: push the value StackIn onto the top of the stack.
  - 3: pop the top two values from the stack
  - 4: pop the top to values and then push the value StackIn onto the top of the stack.

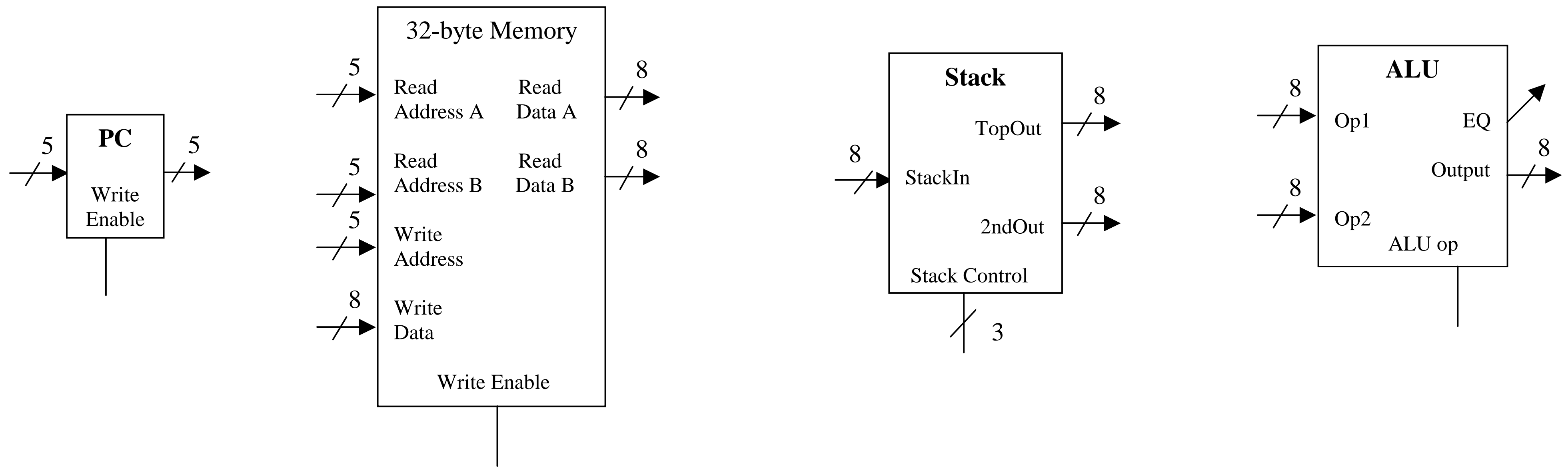
  As with the other registered devices, any changes to the stack don't occur until the end of the clock cycle.



- **Sign extend:** This device sign extends a 5-bit 2's complement number into an 8-bit 2's complement number.

Using the above devices plus wires and Muxes, create a single-cycle data path for the HOB ISA. You may drive constant values onto the wires if needed and should use as little hardware as possible. We have supplied two copies of a template for your answer. You may need to add additional devices from the above list. *You must clearly indicate which copy of the template you want us to grade*. **[22 points]**


PC
Write Enable
5
5
32-byte Memory
Read Address A
Read Address B
Write Address
Write Data
Read Data A
Read Data B
Write Enable
5
5
5
8
8
8
Stack
StackIn
TopOut
2ndOut
Stack Control
8
8
8
3
ALU
Op1
Op2
EQ
Output
ALU op
8
8
8


PC
5
5
Write
Enable
32-byte Memory
5
Read
Address A
Read
Data A
8
5
Read
Address B
Read
Data B
8
5
Write
Address
8
Write
Data
Write Enable
Stack
8
StackIn
TopOut
8
2ndOut
8
Stack Control
3
ALU
8
Op1
EQ
8
Op2
Output
8
ALU op

4. Label each of the MUXes you have added. Create a table which shows the control values for those MUXes and all of the other devices used when performing a beq instruction. **[10 points]**